MULTI-AGENT SYSTEMS

MULTI-AGENT SYSTEMS

MULTI-AGENT GAMES

Rational and Convergent Learning in Stochastic Games

Michael Bowling mhb@cs.cmu.edu
Manuela Veloso veloso@cs.cmu.edu

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

This paper investigates the problem of policy learning in multiagent environments using the stochastic game framework, which we briefly overview. We introduce two properties as desirable for a learning agent when in the presence of other learning agents, namely rationality and convergence. We examine existing reinforcement learning algorithms according to these two properties and notice that they fail to simultaneously meet both criteria. We then contribute a new learning algorithm, WoLF policy hill-climbing, that is based on a simple principle: "learn quickly while losing, slowly while winning." The algorithm is proven to be rational and we present empirical results for a number of stochastic games showing the algorithm converges.

1 Introduction

The *multiagent learning problem* consists of devising a learning algorithm for *our* single agent to learn a policy in the presence of *other* learning agents that are outside of our control. Since the other agents are also adapting, learning in the presence of multiple learners can be viewed as a problem of a "moving target," where the optimal policy may be changing while we learn. Multiple approaches to multiagent learning have been pursued with different degrees of success (as surveyed in [Weiß and Sen, 1996] and [Stone and Veloso, 2000]). Previous learning algorithms either converge to a policy that is not optimal with respect to the other player's policies, or they may not converge at all. In this paper we contribute an algorithm to overcome these shortcomings.

We examine the multiagent learning problem using the framework of stochastic games. Stochastic games (SGs) are a very natural multiagent extension of Markov decision processes (MDPs), which have been studied extensively as a model of single agent learning. Reinforcement learning [Sutton and Barto, 1998] has been successful at finding optimal control policies in the MDP framework, and has also been examined as the basis for learning in stochastic games [Claus and Boutilier, 1998; Hu and Wellman, 1998; Littman, 1994]. Additionally, SGs have a rich background in game theory, being first introduced in 1953 [Shapley].

In Section 2 we provide a rudimentary review of the necessary game theory concepts: stochastic games, best-responses, and Nash equilibria. In Section 3 we present two desirable properties, rationality and convergence, that help to elucidate the shortcomings of previous algorithms. In Section 4 we contribute a new algorithm toward achieving these properties called WoLF ("Win or Learn Fast") policy hill-climbing, and prove that this algorithm is rational. Finally, in Section 5 we present empirical results of the convergence of this algorithm in a number and variety of domains.

2 Stochastic Games

A *stochastic game* is a tuple $(n, \mathcal{S}, \mathcal{A}_{1\ldots n}, T, R_{1\ldots n})$, where n is the number of agents, $\mathcal{S}$ is a set of states, $\mathcal{A}_i$ is the set of actions available to agent i with $\mathcal{A}$ being the joint action space $\mathcal{A}_1 \times \ldots \times \mathcal{A}_n$, T is a transition function $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and R_i is a reward function for the ith agent $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$. This is very similar to the MDP framework except we have multiple agents selecting actions and the next state and rewards depend on the joint action of the agents. Also notice that each agent has its own separate reward function. The goal for each agent is to select actions in order to maximize its discounted future reward with discount factor γ.

SGs are a very natural extension of MDPs to multiple agents. They are also an extension of matrix games to multiple states. Two common matrix games are in Figure 1. In these games there are two players; one selects a row and the other selects a column of the matrix. The entry of the matrix they jointly select determines the payoffs. The games in Figure 1 are zero-sum games, where the row player receives the payoff in the matrix, and the column player receives the negative of that payoff. In the general case (general-sum games) each player has a separate matrix that determines its payoff.

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$
Matching Pennies

$$\begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$
Rock-Paper-Scissors

Figure 1: Two example matrix games.

Each state in a stochastic game can be viewed as a matrix game with the payoffs for each joint action determined by the matrix entries $R_i(s, a)$. After playing the matrix game and

receiving their payoffs the players are transitioned to another state (or matrix game) determined by their joint action. We can see that SGs then contain both MDPs and matrix games as subsets of the framework.

Mixed Policies. Unlike in single-agent settings, deterministic policies in multiagent settings can often be exploited by the other agents. Consider the matching pennies matrix game as shown in Figure 1. If the column player were to play either action deterministically, the row player could win a payoff of one every time. This requires us to consider mixed strategies or policies. A mixed policy, $\rho : \mathcal{S} \rightarrow PD(\mathcal{A}_i)$, is a function that maps states to mixed strategies, which are probability distributions over the player's actions.

Nash Equilibria. Even with the concept of mixed strategies there are still no optimal strategies that are independent of the other players' strategies. We can, though, define a notion of best-response. A strategy is a *best-response* to the other players' strategies if it is optimal given their strategies. The major advancement that has driven much of the development of matrix games, game theory, and even stochastic games is the notion of a best-response equilibrium, or *Nash equilibrium* [Nash, Jr., 1950].

A Nash equilibrium is a collection of strategies for each of the players such that each player's strategy is a best-response to the other players' strategies. So, no player can get a higher payoff by changing strategies given that the other players also don't change strategies. What makes the notion of equilibrium compelling is that all matrix games have such an equilibrium, possibly having multiple equilibria. In the zero-sum examples in Figure 1, both games have an equilibrium consisting of each player playing the mixed strategy where all the actions have equal probability.

The concept of equilibria also extends to stochastic games. This is a non-trivial result, proven by Shapley [1953] for zero-sum stochastic games and by Fink [1964] for general-sum stochastic games.

3 Motivation

The multiagent learning problem is one of a "moving target." The best-response policy changes as the other players, which are outside of our control, change their policies. Equilibrium solutions do not solve this problem since the agent does not know which equilibrium the other players will play, or even if they will tend to an equilibrium at all.

Devising a learning algorithm for our agent is also challenging because we don't know which *learning algorithms* the other learning agents are using. Assuming a general case where other players may be changing their policies in a completely arbitrary manner is neither useful nor practical. On the other hand, making restrictive assumptions on the other players' specific methods of adaptation is not acceptable, as the other learners are outside of our control and therefore we don't know which restrictions to assume.

We address this multiagent learning problem by defining two properties of a learner that make requirements on its behavior in concrete situations. After presenting these properties we examine previous multiagent reinforcement learning techniques showing that they fail to simultaneously achieve these properties.

3.1 Properties

We contribute two desirable properties of multiagent learning algorithms: rationality and convergence.

Property 1 *(Rationality) If the other players' policies converge to stationary policies then the learning algorithm will converge to a policy that is a best-response to their policies.*

This is a fairly basic property requiring the player to behave optimally when the other players play stationary strategies. This requires the player to learn a best-response policy in this case where one indeed exists. Algorithms that are not rational often opt to learn some policy independent of the other players' policies, such as their part of some equilibrium solution. This completely fails in games with multiple equilibria where the agents cannot *independently select* and play an equilibrium.

Property 2 *(Convergence) The learner will necessarily converge to a stationary policy. This property will usually be conditioned on the other agents using an algorithm from some class of learning algorithms.*

The second property requires that, against some class of other players' learning algorithms (ideally a class encompassing most "useful" algorithms), the learner's policy will converge. For example, one might refer to convergence with respect to players with stationary policies, or convergence with respect to rational players.

In this paper, we focus on convergence in the case of self-play. That is, if all the players use the same learning algorithm do the players' policies converge? This is a crucial and difficult step towards convergence against more general classes of players. In addition, ignoring the possibility of self-play makes the naive assumption that other players are inferior since they cannot be using an identical algorithm.

In combination, these two properties guarantee that the learner will converge to a stationary strategy that is optimal given the play of the other players. There is also a connection between these properties and Nash equilibria. When all players are rational, if they converge, then they must have converged to a Nash equilibrium. Since all players converge to a stationary policy, each player, being rational, must converge to a best response to their policies. Since this is true of each player, then their policies by definition must be an equilibrium. In addition, if all players are rational and convergent with respect to the other players' algorithms, then convergence to a Nash equilibrium is guaranteed.

3.2 Other Reinforcement Learners

There are few RL techniques that directly address learning in a multiagent system. We examine three RL techniques: single-agent learners, joint-action learners (JALs), and minimax-Q.

Single-Agent Learners. Although not truly a multiagent learning algorithm, one of the most common approaches is to apply a single-agent learning algorithm (e.g. Q-learning, TD(λ), prioritized sweeping, etc.) to a multi-agent domain.

They, of course, ignore the existence of other agents, assuming their rewards and the transitions are Markovian. They essentially treat other agents as part of the environment.

This naive approach does satisfy one of the two properties. If the other agents play, or converge to, stationary strategies then their Markovian assumption holds and they converge to an optimal response. So, single agent learning is rational. On the other hand, it is not generally convergent in self-play. This is obvious to see for algorithms that learn only deterministic policies. Since they are rational, if they converge it must be to a Nash equilibrium. In games where the only equilibria are mixed equilibria (e.g. Matching Pennies), they could not converge. There are single-agent learning algorithms capable of playing stochastic policies [Jaakkola *et al.*, 1994; Baird and Moore, 1999]. In general though just the ability to play stochastic policies is not sufficient for convergence, as will be shown in Section 4.

Joint Action Learners. JALs [Claus and Boutilier, 1998] observe the actions of the other agents. They assume the other players are selecting actions based on a stationary policy, which they estimate. They then play optimally with respect to this learned estimate. Like single-agent learners they are rational but not convergent, since they also cannot converge to mixed equilibria in self-play.

Minimax-Q. Minimax-Q [Littman, 1994] and Hu & Wellman's extension of it to general-sum SGs [1998] take a different approach. These algorithms observe both the actions and rewards of the other players and try to learn a Nash equilibrium explicitly. The algorithms learn and play the equilibrium independent of the behavior of other players. These algorithms are convergent, since they always converge to a stationary policy. However, these algorithms are not rational. This is most obvious when considering a game of Rock-Paper-Scissors against an opponent that almost always plays "Rock". Minimax-Q will still converge to the equilibrium solution, which is not optimal given the opponent's policy.

In this work we are looking for a learning technique that is rational, and therefore plays a best-response in the obvious case where one exists. Yet, its policy should still converge. We want the rational behavior of single-agent learners and JALs, and the convergent behavior of minimax-Q.

4 A New Algorithm

In this section we contribute an algorithm towards the goal of a rational and convergent learner. We first introduce an algorithm that is rational and capable of playing mixed policies, but does not converge in experiments. We then introduce a modification to this algorithm that results in a rational learner that does in experiments converge to mixed policies.

4.1 Policy Hill Climbing

A simple extension of Q-learning to play mixed strategies is policy hill-climbing (PHC) as shown in Table 1. The algorithm, in essence, performs hill-climbing in the space of mixed policies. Q-values are maintained just as in normal Q-learning. In addition the algorithm maintains the current mixed policy. The policy is improved by increasing the probability that it selects the highest valued action according to a learning rate $\delta \in (0, 1]$. Notice that when $\delta = 1$ the algorithm is equivalent to Q-learning, since with each step the policy moves to the greedy policy executing the highest valued action with probability 1 (modulo exploration).

1. Let α and δ be learning rates. Initialize,
$$Q(s,a) \leftarrow 0, \qquad \pi(s,a) \leftarrow \frac{1}{|\mathcal{A}_i|}.$$
2. Repeat,
   (a) From state s select action a with probability $\pi(s,a)$ with some exploration.
   (b) Observing reward r and next state s',
   $$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a')\right).$$
   (c) Update $\pi(s,a)$ and constrain it to a legal probability distribution,
   $$\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta & \text{if } a = \text{argmax}_{a'} Q(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases}.$$

Table 1: Policy hill-climbing algorithm (PHC) for player i.

This technique, like Q-learning, is rational and will converge to an optimal policy if the other players are playing stationary strategies. The proof follows from the proof of Q-learning, which guarantees the Q values will converge to Q^* with a suitable exploration policy.[1] Similarly, π will converge to a policy that is greedy according to Q, which is converging to Q^*, the optimal response Q-values. Despite the fact that it is rational and can play mixed policies, it still doesn't show any promise of being convergent. We show examples of its convergence failures in Section 5.

4.2 WoLF Policy Hill-Climbing

We now introduce the main contribution of this paper. The contribution is two-fold: using a *variable learning rate*, and the *WoLF principle*. We demonstrate these ideas as a modification to the naive policy hill-climbing algorithm.

The basic idea is to vary the learning rate used by the algorithm in such a way as to encourage convergence, without sacrificing rationality. We propose the WoLF principle as an appropriate method. The principle has a simple intuition, learn quickly while losing and slowly while winning. The specific method for determining when the agent is winning is by comparing the current policy's expected payoff with that of the average policy over time. This principle aids in convergence by giving more time for the other players to adapt to changes in the player's strategy that at first appear beneficial, while allowing the player to adapt more quickly to other players' strategy changes when they are harmful.

The required changes for WoLF policy hill-climbing are shown in Table 2. Practically, the algorithm requires two

[1] The issue of exploration is not critical to this work. See [Singh *et al.*, 2000a] for suitable exploration policies for online learning.

1. Let α, $\delta_l > \delta_w$ be learning rates. Initialize,

$$Q(s,a) \leftarrow 0, \qquad \pi(s,a) \leftarrow \frac{1}{|\mathcal{A}_i|}, \qquad C(s) \leftarrow 0.$$

2. Repeat,

(a,b) Same as PHC in Table 1

(c) Update estimate of average policy, $\bar{\pi}$,

$$\begin{array}{rcl} C(s) & \leftarrow & C(s)+1 \\ \forall a' \in \mathcal{A}_i \quad \bar{\pi}(s,a') & \leftarrow & \bar{\pi}(s,a') + \\ & & \frac{1}{C(s)}\left(\pi(s,a') - \bar{\pi}(s,a')\right). \end{array}$$

(d) Update $\pi(s,a)$ and constrain it to a legal probability distribution,

$$\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta & \text{if } a = \text{argmax}_{a'} Q(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases},$$

where,

$$\delta = \begin{cases} \delta_w & \text{if } \sum_a \pi(s,a)Q(s,a) > \sum_a \bar{\pi}(s,a)Q(s,a) \\ \delta_l & \text{otherwise} \end{cases}.$$

Table 2: WoLF policy hill-climbing algorithm for player i.

learning learning rate parameters, δ_l and δ_w, with $\delta_l > \delta_w$. The learning rate that is used to update the policy depends on whether the agent is currently winning (δ_w) or losing (δ_l). This is determined by comparing the expected value, using the current Q-value estimates, of following the current policy π in the current state with that of following the average policy $\bar{\pi}$. If the expectation of the current policy is smaller (i.e. the agent is "losing") then the larger learning rate, δ_l is used.

WoLF policy hill-climbing remains rational, since only the speed of learning is altered. In fact, any bounded variation of the learning rate would retain rationality. Its convergence properties, though, are quite different. In the next section we show empirical results that this technique converges to rational policies for a number and variety of stochastic games. The WoLF principle also has theoretical justification for a restricted class of games. For two-player, two-action, iterated matrix games, gradient ascent (which is known not to converge [Singh *et al.*, 2000b]) when using a WoLF varied learning rate is guaranteed to converge to a Nash equilibrium in self-play [Bowling and Veloso, 2001].

Something similar to the WoLF principle has also been studied in some form in other areas, notably when considering an adversary. In evolutionary game theory the *adjusted replicator dynamics* [Weibull, 1995] scales the individuals' growth rate by the inverse of the overall success of the population. This will cause the population's composition to change more quickly when the population as a whole is performing poorly. A form of this also appears as a modification to the *randomized weighted majority* algorithm [Blum and Burch, 1997]. In this algorithm, when an expert makes a mistake, a portion of its weight loss is redistributed among the other experts. If the algorithm is placing large weights on mistaken experts (i.e. the algorithm is "losing"), then a larger portion of the weights are redistributed (i.e. the algorithm adapts more quickly.) Neither research lines recognized their modification as essentially involving a variable learning rate, nor has such an approach been applied to learning in stochastic games.

5 Results

In this section we show results of applying policy hill-climbing and WoLF policy hill-climbing to a number of different games, from the multiagent reinforcement learning literature. The domains include two matrix games that help to show how the algorithms work and the effect of the WoLF principle on convergence. The algorithms were also applied to two multi-state SGs. One is a general-sum grid world domain used by Hu & Wellman [1998]. The other is a zero-sum soccer game introduced by Littman [1994].

The experiments involve training the players using the same learning algorithm. Since PHC and WoLF-PHC are rational, we know that if they converge against themselves, then they must have converged to a Nash equilibrium. For the matrix game experiments $\delta_l/\delta_w = 2$, but for the other results a more aggressive $\delta_l/\delta_w = 4$ was used. In all cases both the δ and α were decreased proportionately to $1/C(s)$, although the exact proportion varied between domains.

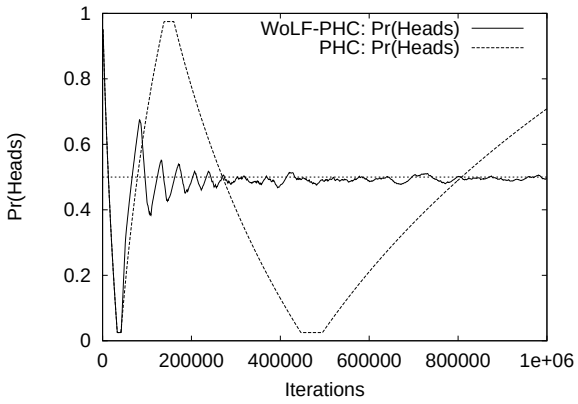
5.1 Matrix Games

The algorithms were applied to the two matrix games, from Figure 1. In both games, the Nash equilibrium is a mixed policy consisting of executing the actions with equal probability. The large number of trials and small ratio of the learning rates were used for the purpose of illustrating how the algorithm learns and converges.

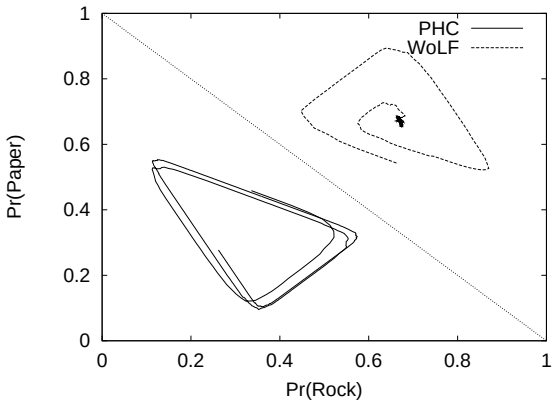
The results of applying both policy hill-climbing and WoLF policy hill-climbing to the matching pennies game is shown in Figure 2(a). WoLF-PHC quickly begins to oscillate around the equilibrium, with ever decreasing amplitude. On the other hand, PHC oscillates around the equilibrium but without any appearance of converging. This is even more obvious in the game of rock-paper-scissors. The results are shown in Figure 2(b), and show trajectories of the players' strategies in policy space through one million steps. Policy hill-climbing circles the equilibrium policy without any hint of converging, while WoLF policy hill-climbing very nicely spirals towards the equilibrium.

5.2 Gridworld

We also examined a gridworld domain introduced by Hu and Wellman [1998] to demonstrate their extension of Minimax-Q to general-sum games. The game consists of a small grid shown in Figure 3(a). The agents start in two corners and are trying to reach the goal square on the opposite wall. The players have the four compass actions (i.e. N, S, E, and W), which are in most cases deterministic. If the two players attempt to move to the same square, both moves fail. To make the game interesting and force the players to interact, while in the initial starting position the North action is uncertain, and is only executed with probability 0.5. The optimal path for each agent is to move laterally on the first move and then move North to the goal, but if both players move laterally then the actions will fail. There are two Nash equilibria for this game. They involve one player taking the lateral move



(a) Matching Pennies Game



(b) Rock-Paper-Scissors Game

Figure 2: (a) Results for matching pennies: the policy for one of the players as a probability distribution while learning with PHC and WoLF-PHC. The other player's policy looks similar. (b) Results for rock-paper-scissors: trajectories of one player's policy. The bottom-left shows PHC in self-play, and the upper-right shows WoLF-PHC in self-play.

and the other trying to move North. Hence the game requires that the players coordinate their behaviors.

WoLF policy hill-climbing successfully converges to one of these equilibria. Figure 3(a) shows an example trajectory of the players' strategies for the initial state while learning over 100,000 steps. In this example the players converged to the equilibrium where player one moves East and player two moves North from the initial state. This is evidence that WoLF policy hill-climbing can learn an equilibrium even in a general-sum game with multiple equilibria.

5.3 Soccer

The final domain is a comparatively large zero-sum soccer game introduced by Littman [1994] to demonstrate Minimax-Q. An example of an initial state in this game is shown in Figure 3(b), where player 'B' has possession of the ball. The goal is for the players to carry the ball into the goal on the opposite side of the field. The actions available are the four compass directions and the option to not move. The players select actions simultaneously but they are executed in a random order, which adds non-determinism to their actions. If a player attempts to move to the square occupied by its opponent, the stationary player gets possession of the ball, and the move fails. Unlike the grid world domain, the Nash equilibrium for this game requires a mixed policy. In fact any deterministic policy (therefore anything learned by an single-agent learner or JAL) can always be defeated [Littman, 1994].

Our experimental setup resembles that used by Littman in order to compare with his results for Minimax-Q. Each player was trained for one million steps. After training, its policy was fixed and a challenger using Q-learning was trained against the player. This determines the learned policy's worst-case performance, and gives an idea of how close the player was to the equilibrium policy, which would perform no worse than losing half its games to its challenger. Unlike Minimax-Q, WoLF-PHC and PHC generally oscillate around the target solution. In order to account for this in the results, training was continued for another 250,000 steps and evaluated after every 50,000 steps. The *worst performing policy* was then used for the value of that learning run.

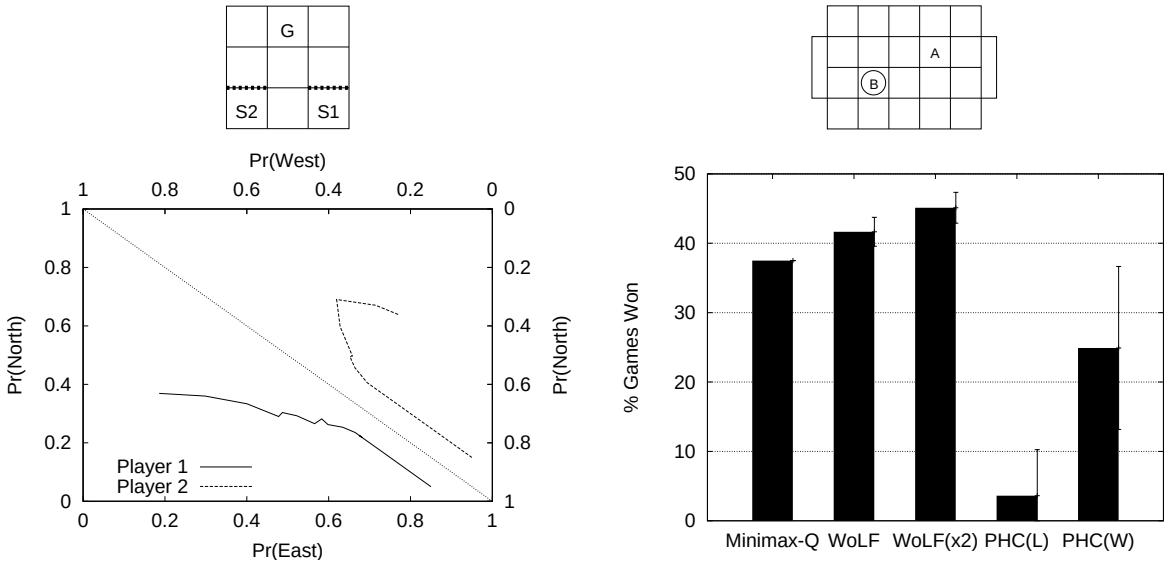
Figure 3(b) shows the percentage of games won by the different players when playing their challengers. "Minimax-Q" represents Minimax-Q when learning against itself (the results were taken from Littman's original paper.) "WoLF" represents WoLF policy hill-climbing learning against itself. "PHC(L)" and "PHC(W)" represents policy hill-climbing with $\delta = \delta_l$ and $\delta = \delta_w$, respectively. "WoLF(2x)" represents WoLF policy hill-climbing learning with twice the training (i.e. two million steps). The performance of the policies were averaged over fifty training runs and the standard deviations are shown by the lines beside the bars. The relative ordering by performance is statistically significant.

WoLF-PHC does extremely well, performing equivalently to Minimax-Q with the same amount of training[2] and continues to improve with more training. The exact effect of the WoLF principle can be seen by its out-performance of PHC, using either the larger or smaller learning rate. This shows that the success of WoLF-PHC is not simply due to changing learning rates, but rather to changing the learning rate at the appropriate time to encourage convergence.

6 Conclusion

In this paper we present two properties, rationality and convergence, that are desirable for a multiagent learning algorithm. We present a new algorithm that uses a variable learning rate based on the WoLF ("Win or Learn Fast") principle. We then showed how this algorithm takes large steps towards achieving these properties on a number and variety of stochastic games. The algorithm is rational and is shown empirically to converge in self-play to an equilibrium even in games with multiple or mixed policy equilibria, which previous multiagent reinforcement learners have not achieved.

[2]The results are not directly comparable due to the use of a different decay of the learning rate. Minimax-Q uses an exponential decay that decreases too quickly for use with WoLF-PHC.



(a) Gridworld Game (b) Soccer Game

Figure 3: (a) Gridworld game. The dashed walls represent the actions that are uncertain. The results show trajectories of two players' policies while learning with WoLF-PHC. (b) Soccer game. The results show the percentage of games won against a specifically trained worst-case opponent after one million steps of training.

Acknowledgements. Thanks to Will Uther for ideas and discussions. This research was sponsored by the United States Air Force under Grants Nos F30602-00-2-0549 and F30602-98-2-0135. The content of this publication does not necessarily reflect the position or the policy of the sponsors and no official endorsement should be inferred.

References

[Baird and Moore, 1999] L. C. Baird and A. W. Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*. The MIT Press, 1999.

[Blum and Burch, 1997] A. Blum and C. Burch. On-line learning and the metrical task system problem. In *Tenth Annual Conference on Computational Learning Theory*, 1997.

[Bowling and Veloso, 2001] M. Bowling and M. Veloso. Convergence of gradient dynamics with a variable learning rate. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001. To Appear.

[Claus and Boutilier, 1998] C. Claus and C. Boutilier. The dyanmics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. AAAI Press, 1998.

[Fink, 1964] A. M. Fink. Equilibrium in a stochastic n-person game. *Journal of Science in Hiroshima University, Series A-I*, 28:89–93, 1964.

[Hu and Wellman, 1998] J. Hu and M. P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 242–250, 1998.

[Jaakkola *et al.*, 1994] T. Jaakkola, S. P. Singh, and M. I. Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *Advances in Neural Information Processing Systems 6*, 1994.

[Littman, 1994] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, 1994.

[Nash, Jr., 1950] J. F. Nash, Jr. Equilibrium points in n-person games. *PNAS*, 36:48–49, 1950.

[Shapley, 1953] L. S. Shapley. Stochastic games. *PNAS*, 39:1095–1100, 1953.

[Singh *et al.*, 2000a] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 2000.

[Singh *et al.*, 2000b] S. Singh, M. Kearns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 541–548, 2000.

[Stone and Veloso, 2000] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3), 2000.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. The MIT Press, 1998.

[Weibull, 1995] J. W. Weibull. *Evolutionary Game Theory*. The MIT Press, 1995.

[Weiß and Sen, 1996] G. Weiß and S. Sen, editors. *Adaptation and Learning in Multiagent Systems*. Springer, 1996.

Multi-Agent Influence Diagrams for Representing and Solving Games

Daphne Koller
Computer Science Dept.
Stanford University
Stanford, CA 94305-9010
koller@cs.stanford.edu

Brian Milch
Computer Science Dept.
Stanford University
Stanford, CA 94305-9010
milch@cs.stanford.edu

Abstract

The traditional representations of games using the extensive form or the strategic (normal) form obscure much of the structure that is present in real-world games. In this paper, we propose a new representation language for general multi-player games — *multi-agent influence diagrams (MAIDs)*. This representation extends graphical models for probability distributions to a multi-agent decision-making context. MAIDs explicitly encode structure involving the dependence relationships among variables. As a consequence, we can define a notion of *strategic relevance* of one decision variable to another: D' is *strategically relevant* to D if, to optimize the decision rule at D, the decision maker needs to take into consideration the decision rule at D'. We provide a sound and complete graphical criterion for determining strategic relevance. We then show how strategic relevance can be used to detect structure in games, allowing a large game to be broken up into a set of interacting smaller games, which can be solved in sequence. We show that this decomposition can lead to substantial savings in the computational cost of finding Nash equilibria in these games.

1 Introduction

Game theory [Fudenberg and Tirole, 1991] provides a mathematical framework for determining what behavior is rational for agents interacting with each other in a partially observable environment. However, the traditional representations of games are primarily designed to be amenable to abstract mathematical formulation and analysis. As a consequence, the standard game representations, both the normal (matrix) form and the extensive (game tree) form, obscure certain important structure that is often present in real-world scenarios — the decomposition of the situation into chance and decision *variables*, and the dependence relationships between these variables. In this paper, we provide a representation that captures this type of structure. We also show that capturing this structure explicitly has several advantages, both in our ability to analyze the game in novel ways, and in our ability to compute Nash equilibria efficiently.

Our framework of *multi-agent influence diagrams (MAIDs)* extends the formalisms of *Bayesian networks* (BNs) [Pearl, 1988] and *influence diagrams* [Howard and Matheson, 1984] to represent decision problems involving multiple agents. MAIDs have clearly defined semantics as noncooperative games: a MAID can be reduced to an equivalent game tree, albeit at the cost of obscuring the variable-level interaction structure that the MAID makes explicit. MAIDs allow us to describe complex games using a natural representation, whose size is no larger than that of the extensive form, but which can be exponentially more compact.

Just as Bayesian networks make explicit the dependencies between probabilistic variables, MAIDs make explicit the dependencies between *decision* variables. They allow us to define a qualitative notion of *strategic relevance*: a decision variable D strategically relies on another decision variable D' when, to optimize the decision rule at D, the decision-making agent needs to take into consideration the decision rule at D'. This notion provides new insight about the relationships between the agents' decisions in a strategic interaction. We provide a graph-based criterion, which we call *s-reachability*, for determining strategic relevance based purely on the graph structure, and show that it is sound and complete in the same sense that d-separation is sound and complete for probabilistic dependence. We also provide a polynomial time algorithm for computing s-reachability.

The notion of strategic relevance allows us to define a data structure that we call the *relevance graph* — a directed graph that indicates when one decision variable in the MAID relies on another. We show that this data structure can be used to provide a natural decomposition of a complex game into interacting fragments, and provide an algorithm that finds equilibria for these smaller games in a way that is guaranteed to produce a global equilibrium for the entire game. We show that our algorithm can be exponentially more efficient than an application of standard game-theoretic solution algorithms, including the more efficient solution algorithms of [Romanovskii, 1962; Koller *et al.*, 1994] that work directly on the game tree.

2 Multi-Agent Influence Diagrams (MAIDs)

We will introduce MAIDs using a simple two-agent scenario:

Example 1 *Alice is considering building a patio behind her house, and the patio would be more valuable to her if she could get a clear view of the ocean. Unfortunately, there is a tree in her neighbor Bob's yard that blocks her view. Being somewhat unscrupulous, Alice considers poisoning Bob's tree, which might cause it to become sick. Bob cannot tell*

whether Alice has poisoned his tree, but he can tell if the tree is getting sick, and he has the option of calling in a tree doctor (at some cost). The attention of a tree doctor reduces the chance that the tree will die during the coming winter. Meanwhile, Alice must make a decision about building her patio before the weather gets too cold. When she makes this decision, she knows whether a tree doctor has come, but she cannot observe the health of the tree directly. A MAID for this scenario is shown in Fig. 1.

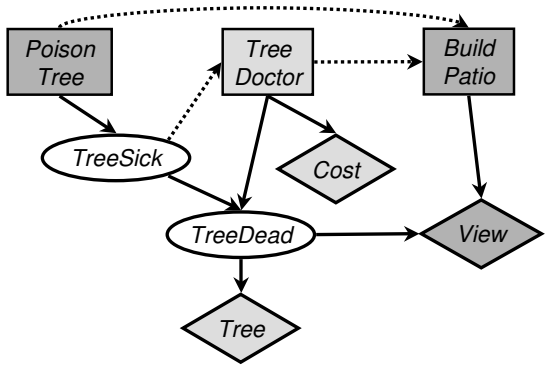


Figure 1: A MAID for the Tree Killer example; Alice's decision and utility variables are in dark gray and Bob's in light gray.

To define a MAID, we begin with a set $\mathcal{A}$ of agents. The world in which the agents act is represented by the set $\mathcal{X}$ of *chance variables*, and a set $\mathcal{D}_a$ of *decision variables* for each agent $a \in \mathcal{A}$. Chance variables correspond to decisions of nature, as in *Bayesian networks* [Pearl, 1988]. They are represented in the diagram as ovals. The decision variables for agent a are variables whose values a gets to choose, and are represented as rectangles in the diagram. We use $\mathcal{D}$ to denote $\bigcup_{a \in \mathcal{A}} \mathcal{D}_a$. The agents' utility functions are specified using *utility variables*: For each agent $a \in \mathcal{A}$, we have a set $\mathcal{U}_a$ of utility variables, represented as diamonds in the diagram. Each variable X has a finite set $dom(X)$ of possible values, called its *domain*. The domain of a utility variable is always a finite set of real numbers (a chance or decision variable can have any finite domain). We use $\mathcal{U}$ to denote $\bigcup_{a \in \mathcal{A}} \mathcal{U}_a$. and $\mathcal{V}$ to denote $\mathcal{X} \cup \mathcal{D} \cup \mathcal{U}$.

Like a BN, a MAID defines a directed acyclic graph with its variables as the nodes, where each variable X is associated with a set of parents $Pa(X) \subset \mathcal{X} \cup \mathcal{D}$. Note that utility variables cannot be parents of other variables. For each chance variable $X \in \mathcal{X}$, the MAID specifies a *conditional probability distribution (CPD)*: a distribution $\Pr(X \mid \mathbf{pa})$ for each instantiation **pa** of $Pa(X)$. For a decision variable $D \in \mathcal{D}_a$, $Pa(D)$ is the set of variables whose values agent a knows when he chooses a value for D. Thus, the choice agent a makes for D can be contingent only on these variables. (See Definition 1 below.) For a utility variable U, the MAID also specifies a CPD $\Pr(U \mid \mathbf{pa})$ for each instantiation **pa** of $Pa(X)$. However, we require that the value of a utility variable be a deterministic function of the values of its parents: for each $\mathbf{pa} \in dom(Pa(U))$, there is one value of U that has probability 1, and all other values of U have probability 0. We use $U(\mathbf{pa})$ to denote the value of node U that has probability 1 when $Pa(U) = \mathbf{pa}$. The total utility that an agent a derives from an instantiation of $\mathcal{V}$ is the sum of the values of $\mathcal{U}_a$ in this instantiation; thus, we are defining an additive decomposition of the agent's utility function.

The agents get to select their behavior at each of their decision nodes. An agent's decision at a variable D can depend on the variables that the agent observes prior to making D — D's parents. The agent's choice of strategy is specified via a set of *decision rules*.

Definition 1 *A* decision rule *for a decision variable D is a function that maps each instantiation **pa** of $Pa(D)$ to a probability distribution over $dom(D)$. An assignment of decision rules to every decision $D \in \mathcal{D}_a$ for a particular agent $a \in \mathcal{A}$ is called a* strategy.

An assignment σ of decision rules to every decision $D \in \mathcal{D}$ is called a *strategy profile*. A *partial strategy profile* $\sigma_{\mathcal{E}}$ is an assignment of decision rules to a subset $\mathcal{E}$ of $\mathcal{D}$. We will also use $\sigma_{\mathcal{E}}$ to denote the restriction of σ to $\mathcal{E}$, and $\sigma_{-\mathcal{E}}$ to denote the restriction of σ to variables not in $\mathcal{E}$.

Note that a decision rule has exactly the same form as a CPD. Thus, if we have a MAID $\mathcal{M}$, then a partial strategy profile $\sigma_{\mathcal{E}}$ that assigns decision rules to a set $\mathcal{E}$ of decision variables induces a new MAID $\mathcal{M}[\sigma_{\mathcal{E}}]$ where the elements of $\mathcal{E}$ have become chance variables. That is, each $D \in \mathcal{E}$ corresponds to a chance variable in $\mathcal{M}[\sigma_{\mathcal{E}}]$ with $\sigma_{\mathcal{E}}(D)$ as its CPD. When σ assigns a decision rule to every decision variable in $\mathcal{M}$, the induced MAID is simply a BN: it has no more decision variables. This BN defines a joint probability distribution $P_{\mathcal{M}[\sigma]}$ over all the variables in $\mathcal{M}$.

Definition 2 *If $\mathcal{M}$ is a MAID and σ is a strategy profile for $\mathcal{M}$, then the* joint distribution for $\mathcal{M}$ induced by σ, *denoted $P_{\mathcal{M}[\sigma]}$, is the joint distribution over $\mathcal{V}$ defined by the Bayes net where:*

- *the set of variables is $\mathcal{V}$;*
- *for $X, Y \in \mathcal{V}$, there is an edge $X \rightarrow Y$ iff $X \in Pa(Y)$;*
- *for all $X \in \mathcal{X} \cup \mathcal{U}$, the CPD for X is $\Pr(X)$;*
- *for all $D \in \mathcal{D}$, the CPD for D is $\sigma(D)$.*

We can now write an equation for the utility that agent a expects to receive in a MAID $\mathcal{M}$ if the agents play a given strategy profile σ. Suppose $\mathcal{U}_a = \{U_1, \ldots, U_m\}$. Then:

$$\mathrm{EU}_a(\sigma) = \sum_{(u_1,\ldots,u_m) \in dom(\mathcal{U}_a)} P_{\mathcal{M}[\sigma]}(u_1, \ldots, u_m) \sum_{i=1}^{m} u_i \tag{1}$$

where $dom(\mathcal{U}_a)$ is the joint domain of $\mathcal{U}_a$.

Because the expectation of a sum of random variables is the same as the sum of the expectations of the individual random variables, we can also write this equation as:

$$\mathrm{EU}_a(\sigma) = \sum_{U \in \mathcal{U}_a} \sum_{u \in dom(U)} P_{\mathcal{M}[\sigma]}(U = u) \cdot u \tag{2}$$

Having defined the notion of an expected utility, we can now define what it means for an agent to optimize his decision at one or more of his decision rules, relative to a given set of decision rules for the other variables.

Definition 3 *Let $\mathcal{E}$ be a subset of $\mathcal{D}_a$, and let σ be a strategy profile. We say that $\sigma^*_{\mathcal{E}}$ is* optimal for the strategy profile σ *if, in the induced MAID $\mathcal{M}[\sigma_{-\mathcal{E}}]$, where the only remaining decisions are those in $\mathcal{E}$, the strategy $\sigma^*_{\mathcal{E}}$ is optimal, i.e., for all strategies $\sigma'_{\mathcal{E}}$:*

$$EU_a((\sigma_{-\mathcal{E}}, \sigma^*_{\mathcal{E}})) \geq EU_a((\sigma_{-\mathcal{E}}, \sigma'_{\mathcal{E}}))$$

Note that, in this definition, it does not matter what decision rules σ assigns to the variables in $\mathcal{E}$.

In the game-theoretic framework, we typically consider a strategy profile to represent rational behavior if it is a *Nash equilibrium* [Nash, 1950]. Intuitively, a strategy profile is a Nash equilibrium if no agent has an incentive to deviate from the strategy specified for him by the profile, as long as the other agents do not deviate from their specified strategies.

Definition 4 *A strategy profile σ is a* Nash equilibrium *for a MAID $\mathcal{M}$ if for all agents $a \in \mathcal{A}$, $\sigma_{\mathcal{D}_a}$ is optimal for the strategy profile σ.*

3 MAIDs and Games

A MAID provides a compact representation of a scenario that can also be represented as a game in strategic or extensive form. In this section, we discuss how to convert a MAID into an extensive-form game. We also show how, once we have found an equilibrium strategy profile for a MAID, we can convert it into a behavior strategy profile for the extensive form game. The word "node" in this section refers solely to a node in the tree, as distinguished from the nodes in the MAID.

We use a straightforward extension of a construction of [Pearl, 1988] for converting an influence diagram into a decision tree. The basic idea is to construct a tree with splits for decision and chance nodes in the MAID. However, to reduce the exponential blowup, we observe that we do not need to split on every chance variable in the MAID. A chance variable that is never observed by any decision can be eliminated by summing it out in the probability and utility computations. We present the construction below, referring the reader to [Pearl, 1988] for a complete discussion.

The set of variables included in our game tree is $\mathcal{G} = \mathcal{D} \cup \bigcup_{D \in \mathcal{D}} Pa(D)$. We define a total ordering $\prec$ over $\mathcal{G}$ that is consistent with the topological order of the MAID: if there is a directed path from X to Y, then $X \prec Y$. Our tree $\mathcal{T}$ is a symmetric tree, with each path containing splits over all the variables in $\mathcal{G}$ in the order defined by $\prec$. Each node is labeled with a partial instantiation *inst*(N) of $\mathcal{G}$, in the obvious way. For each agent a, the nodes corresponding to variables $D \in \mathcal{D}_a$ are decision nodes for a; the other nodes are all chance nodes. To define the information sets, consider two decision nodes M and M' that correspond to a variable D. We place M and M' into the same information set if and only if *inst*(M) and *inst*(M') assign the same values to $Pa(D)$.

Our next task is to determine the split probabilities at the chance nodes. Consider a chance node N corresponding to a chance variable C. For each value $c \in dom(C)$, let N_c be the child of N corresponding to the choice $C = c$. We want to compute the probability of going from N to N_c. The problem, of course, is that a MAID does not define a full joint probability distribution until decision rules for the agents are selected. It turns out that we can choose an arbitrary fully mixed strategy profile σ for our MAID $\mathcal{M}$ (one where no decision has probability zero), and do inference in the BN $\mathcal{M}[\sigma]$ induced by this strategy profile, by computing

$$P_{\mathcal{M}[\sigma]}(inst(N_c) \mid inst(N)) \tag{3}$$

The value of this expression does not depend on our choice of σ. To see why this is true, note that if we split on a decision variable D before C, then the decision rule σ_D does not affect the computation of $P_{\mathcal{M}[\sigma]}(inst(N_c) \mid inst(N))$, because *inst*$(N)$ includes values for D and all its parents. If we split on D after C, then D cannot be an ancestor of C in the MAID. Also, by the topological ordering of the nodes in the tree, we know that *inst*(N) cannot specify evidence on D or any of its descendants. Therefore, σ_D cannot affect the computation. Hence, the probabilities of the chance nodes are well-defined.

We define the payoffs at the leaves by computing a distribution over the utility nodes, given an instantiation of $\mathcal{G}$. For a leaf N, the payoff for agent a is:

$$\sum_{U \in \mathcal{U}_a} \sum_{u \in dom(U)} P_{\mathcal{M}[\sigma]}(U = u \mid inst(N)) \cdot u \tag{4}$$

We can also show that the value of (4) does not depend on our choice of σ. The basic idea here is that *inst*(N) determines the values of D and $Pa(D)$ for each decision variable D. Hence, the agents' moves and information are all fully determined, and the probabilities with which different actions are chosen in σ are irrelevant. We omit details.

The mapping between MAIDs and trees also induces an obvious mapping between strategy profiles in the different representations. A MAID strategy profile specifies a probability distribution over $dom(D)$ for each pair $(D, \mathbf{pa})$, where $\mathbf{pa}$ is an instantiation of $Pa(D)$. The information sets in the game tree correspond one-to-one with these pairs, and a behavior strategy in the game tree is a mapping from information sets to probability distributions. Clearly the two are equivalent.

Based on this construction, we can now state the following equivalence proposition:

Proposition 1 *Let $\mathcal{M}$ be a MAID and $\mathcal{T}$ be its corresponding game tree. Then for any strategy profile σ, the payoff vector for σ in $\mathcal{M}$ is the same as the payoff vector for σ in $\mathcal{T}$.*

The number of nodes in $\mathcal{T}$ is exponential in the number of decision variables, and in the number of chance variables that are observed during the course of the game. While this blowup is unavoidable in a tree representation, it can be quite significant. In some games, a MAID can be exponentially smaller than the extensive game it corresponds to.

Example 2 *Suppose a road is being built from north to south through undeveloped land, and n agents have purchased plots of land along the road. As the road reaches each agent's plot, the agent needs to choose what to build on his land. His utility depends on what he builds, on some private information about the suitability of his land for various purposes, and on what is built north, south, and across the road from his land. The agent can observe what has already been built immediately to the north of his land (on both sides of the road), but he cannot observe further north; nor can he observe what will be built across from his land or south of it.*

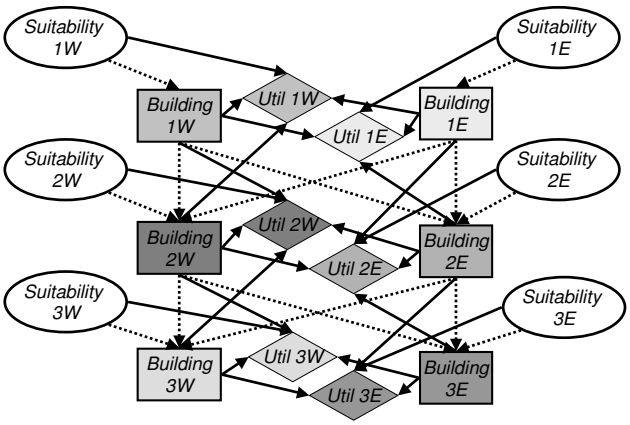


Figure 2: A MAID for the Road example with $n = 6$.

The MAID representation, shown in Fig. 2 for $n = 6$, is very compact. There are n chance nodes, corresponding to the private information about each agent's land, and n decision variables. Each decision variable has at most three parents: the agent's private information, and the two decisions regarding the two plots to the north of the agent's land. Thus, the size of the MAID is linear in n. Conversely, any game tree for this situation must split on each of the n chance nodes and each of the n decisions, leading to a representation that is exponential in n. Concretely, suppose the chance and decision variables each have three possible values, corresponding to three types of buildings. Then the game tree corresponding to the Road *MAID has 3^{2n} leaves.*

A MAID representation is not always more compact. If the game tree is naturally asymmetric, a naive MAID representation can be exponentially larger than the tree. We return to the problem of asymmetric scenarios in Section 6.

4 Strategic Relevance

To take advantage of the independence structure in a MAID, we would like to find a global equilibrium through a series of relatively simple local computations. The difficulty is that, in order to determine the optimal decision rule for a single decision variable, we usually need to know the decision rules for some other variables. In Example 1, when Alice is deciding whether to poison the tree, she needs to compare the expected utilities of her two alternatives. However, the probability of the tree dying depends on the probability of Bob calling a tree doctor if he observes that the tree is sick. Thus, we need to know the decision rule for *CallTreeDoctor* to determine the optimal decision rule for *PoisonTree*. In such situations, we will say that *PoisonTree (strategically) relies on CallTreeDoctor*, or that *CallTreeDoctor* is *relevant to PoisonTree*. On the other hand, *CallTreeDoctor* does not rely on *PoisonTree*. Bob gets to observe whether the tree is sick, and *TreeDead* is conditionally independent of *PoisonTree* given *TreeSick*, so the decision rule for *PoisonTree* is not relevant to Bob's decision.

We will now formalize this intuitive discussion of strategic relevance. Suppose we have a strategy profile, and we would like to find a decision rule for a single decision variable $D \in \mathcal{D}_a$ that maximizes a's expected utility, assuming the rest of the strategy profile remains fixed.

According to Definition 3, to determine whether a decision rule δ for D is optimal for σ, we construct the induced MAID where all decision nodes except D are turned into chance nodes, with their CPDs specified by σ. Then δ is optimal for σ if it maximizes a's expected utility in this single-decision MAID. The key question that motivates our definition of strategic relevance is the following: What other decision rules are relevant for optimizing the decision rule at D?

Definition 5 *Let D be a decision node in a MAID $\mathcal{M}$, δ be a decision rule for D, and σ be a strategy profile such that δ is optimal for σ.* D strategically relies on *a decision node D' in $\mathcal{M}$ if there is another strategy profile σ' such that σ' differs from σ only at D', but δ is not optimal for σ', and neither is any decision rule δ' that agrees with δ on all parent instantiations $\boldsymbol{pa} \in dom(Pa(D))$ where $P_{\mathcal{M}[\sigma]}(\boldsymbol{pa}) > 0$.*

In other words, if a decision rule δ for D is optimal for a strategy profile σ, and D does not rely on D', then δ is also optimal for any strategy profile σ' that differs from σ only at D'. The last clause of this definition is needed to deal with a problem that arises in many other places in game theory — the problem of suboptimal decisions in response to observations that have zero probability (such as observing an irrational move by another agent).

Relevance is a numeric criterion that depends on the specific probabilities and utilities in the MAID. It is not obvious how we would check for strategic relevance without testing all possible pairs of strategy profiles σ and σ'. We would like to find a qualitative criterion which can help us determine strategic relevance purely from the structure of the graph. In other words, we would like to find a criterion which is analogous to the d-separation criterion for determining conditional independence in Bayesian networks.

First, the optimality of the decision rule at D depends only on the utility nodes $\mathcal{U}_D$ that are descendants of D in the MAID. The other utility nodes are irrelevant, because the decision at D cannot influence them. Now, consider another decision variable D'. The decision rule at D' is relevant to D only if it can influence the probability distribution over the utility nodes $\mathcal{U}_D$. To determine whether the CPD for a node can affect the probability distribution over a set of other nodes, we can build on a graphical criterion already defined for Bayesian networks, that of a *requisite probability node*:

Definition 6 *Let G be a BN structure, and let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be sets of variables in the BN. Then a node Z is a* requisite probability node *for the query $P(\boldsymbol{X} \mid \boldsymbol{Y})$ if there exist two Bayesian networks $\mathcal{B}_1$ and $\mathcal{B}_2$ over G, that are identical except in the CPD they assign to Z, but $P_{\mathcal{B}_1}(\boldsymbol{X} \mid \boldsymbol{Y}) \neq P_{\mathcal{B}_2}(\boldsymbol{X} \mid \boldsymbol{Y})$.*

As we will see, the decision rule at D' is only relevant to D if D' (viewed as a chance node) is a requisite probability node for $P(\mathcal{U}_D \mid D, Pa(D))$.

Geiger *et al.* [1990] provide a graphical criterion for testing whether a node Z is a requisite probability node for a query $P(\boldsymbol{X} \mid \boldsymbol{Y})$. We add to Z a new "dummy" parent $\widehat{Z}$ whose

values correspond to CPDs for Z, selected from some set of possible CPDs. Then Z is a requisite probability node for $P(\boldsymbol{X} \mid \boldsymbol{Y})$ if and only if $\widehat{Z}$ can influence $\boldsymbol{X}$ given $\boldsymbol{Y}$.

Based on these considerations, we can define *s-reachability*, a graphical criterion for detecting strategic relevance. Note that unlike d-separation in Bayesian networks, s-reachability is not necessarily a symmetric relation.

Definition 7 *A node D' is* s-reachable *from a node D in a MAID $\mathcal{M}$ if there is some utility node $U \in \mathcal{U}_D$ such that if a new parent $\widehat{D'}$ were added to D', there would be an active path in $\mathcal{M}$ from $\widehat{D'}$ to U given $Pa(D) \cup \{D\}$, where a path is active in a MAID if it is active in the same graph, viewed as a BN.*

We can show that s-reachability is sound and complete for strategic relevance (almost) in the same sense that d-separation is sound and complete for independence in Bayesian networks. As for d-separation, the soundness result is very strong: without s-reachability, one decision cannot be relevant to another.

Theorem 1 (Soundness) *If D and D' are two decision nodes in a MAID $\mathcal{M}$ and D' is not s-reachable from D in $\mathcal{M}$, then D does not rely on D'.*

As for BNs, the result is not as strong in the other direction: s-reachability does not imply relevance in *every* MAID. We can choose the probabilities and utilities in the MAID in such a way that the influence of one decision rule on another does not manifest itself. However, s-reachability is the most precise graphical criterion we can use: it will not identify a strategic relevance unless that relevance actually exists in some MAID that has the given graph structure. We say that two MAIDs have the same graph structure when the two MAIDs have the same sets of variables and agents, each variable has the same parents in the two MAIDs, and the assignment of decision and utility variables to agents is the same in both MAIDs. The chance and decision variables must have the same domains in both MAIDs, but we allow the actual utility values of the utility variables (their domains) to vary. The CPDs in the two MAIDS may also be different.

Theorem 2 (Completeness) *If a node D' is s-reachable from a node D in a MAID, then there is some MAID with the same graph structure in which D relies on D'.*

Since s-reachability is a binary relation, we can represent it as a directed graph. As we show below, this graph turns out to be extremely useful.

Definition 8 *The* relevance graph *for a MAID $\mathcal{M}$ is a directed graph whose nodes are the decision nodes of $\mathcal{M}$, and which contains an edge $D \rightarrow D'$ if and only if D' is s-reachable from D.*

The relevance graph for the Tree Killer example is shown in Fig. 4(a). By Theorem 1, if D relies on D', then there is an edge from D to D' in the relevance graph.

To construct the graph for a given MAID, we need to determine, for each decision node D, the set of nodes D' that are s-reachable from D. Using an algorithm such as Shachter's Bayes-Ball [Shachter, 1998], we can find this set for any given D in time linear in the number of nodes in the MAID.

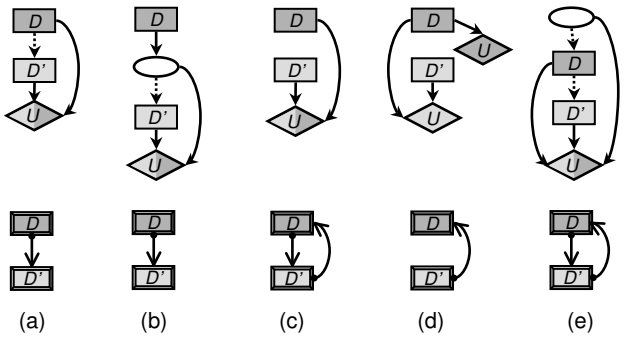


Figure 3: Five simple MAIDs (top), and their relevance graphs (bottom). A two-color diamond represents a pair of utility nodes, one for each agent, with the same parents.

By repeating the algorithm for each D, we can derive the relevance graph in time quadratic in the number of MAID nodes.

Recall our original statement that a decision node D strategically relies on a decision node D' if one needs to know the decision rule for D' in order to evaluate possible decision rules for D. Although we now have a graph-theoretic characterization of strategic relevance, it will be helpful to develop some intuition by examining some simple MAIDs, and seeing when one decision node relies on another. In the five examples shown in Fig. 3, the decision node D belongs to agent a, and D' belongs to agent b. Example (a) represents a perfect-information game. Since agent b can observe the value of D, he does not need to know the decision rule for D in order to evaluate his options. Thus, D' does not rely on D. On the other hand, agent a cannot observe D' when she makes decision D, and D' is relevant to a's utility, so D relies on D'. Example (b) represents a game where the agents do not have perfect information: agent b cannot observe D when making decision D'. However, the information is "perfect enough": the utility for b does not depend on D directly, but only on the chance node, which b can observe. Hence D' does not rely on D. Examples (c) and (d) represent scenarios where the agents move simultaneously, and thus neither can observe the other's move. In (c), each agent's utility node is influenced by both decisions, so D relies on D' and D' relies on D. Thus, the relevance graph is cyclic. In (d), however, the relevance graph is acyclic despite the fact that the agents move simultaneously. The difference here is that agent a no longer cares what agent b does, because her utility is not influenced by b's decision. In graphical terms, there is no active path from D' to a's utility node given D.

One might conclude that a decision node D' never relies on a decision node D when D is observed by D', but the situation is more subtle. Consider example (e), which represents a simple card game: agent a observes a card, and decides whether to bet (D); agent b observes only agent a's bet, and decides whether to bet (D'); the utility of both depends on their bets and the value of the card. Even though agent b observes the actual decision in D, he needs to know the decision rule for D in order to know what the value of D tells him about the chance node. Thus, D' relies on D; indeed,

when D is observed, there is an active path from D that runs through the chance node to the utility node.

5 Computing Equilibria

The computation of a Nash equilibrium for a game is arguably the key computational task in game theory. In this section, we show how the structure of the MAID can be exploited to provide efficient algorithms for finding equilibria in certain games. The key insight behind our algorithm is the use of the relevance graph to break up the task of finding an equilibrium into a series of subtasks, each over a much smaller game. Since algorithms for finding equilibria in general games have complexity that is superlinear in the number of levels in the game tree, breaking the game into smaller games significantly improves the complexity of finding a global equilibrium.

Our algorithm is a generalization of existing backward induction algorithms for decision trees and perfect information games [Zermelo, 1913] and for influence diagrams [Jensen *et al.*, 1994]. The basic idea is as follows: in order to optimize the decision rule for D, we need to know the decision rule for all decisions D' that are relevant for D. For example, the relevance graph for the Tree Killer example (Fig. 4(a)) shows that to optimize *PoisonTree*, we must first decide on the decision rules for *BuildPatio* and *TreeDoctor*. However, we can optimize *TreeDoctor* without knowing the decision rules for either of the other decision variables. Having decided on the decision rule for *TreeDoctor*, we can now optimize *BuildPatio* and then finally *PoisonTree*.

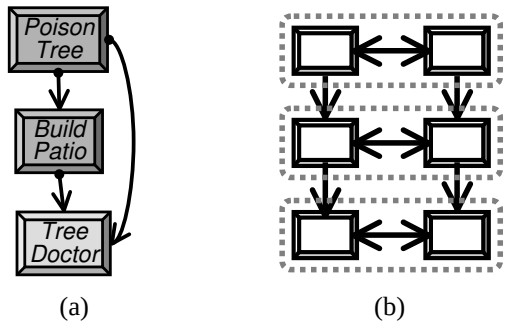


Figure 4: Relevance graphs for (a) the Tree Killer example; (b) the Road example with $n = 6$.

We can apply this simple backward induction procedure in any MAID which, like the Tree Killer example, has an acyclic relevance graph. When the relevance graph is acyclic, we can construct a topological ordering of the decision nodes: an ordering $D_1, \ldots, D_n$ such that if $i < j$, then D_i is not s-reachable from D_j. We can then iterate backward from D_n to D_1, deriving an optimal decision rule for each decision node in turn. Each decision D_i relies only on the decisions that succeed it in the order, and these will have been computed by the time we have to select the decision rule for D_i.

The relevance graph is acyclic in all perfect-information games, and in all single-agent decision problems with perfect recall. There are also some games of imperfect information, such as the Tree Killer example, that have acyclic relevance graphs. But in most games we will encounter cycles in the relevance graph. Consider, for example, any simple two-player simultaneous move game with two decisions D_1 and D_2, where both players' payoffs depend on the decisions at both D_1 and D_2, as in Fig. 3(c). In this case, the optimality of one player's decision rule is clearly intertwined with the other player's choice of decision rule, and the two decision rules must "match" in order to be in equilibrium. Indeed, as we discussed, the relevance graph in such a situation is cyclic.

However, we can often utilize relevance structure even in games where the relevance graph is cyclic.

Example 3 *Consider the relevance graph for the* Road *example, shown in Fig. 4(b) for $n = 6$ agents. We can see that we have pairs of interdependent decision variables, corresponding to the two agents whose lots are across the road from each other. Also, the decision for a given plot relies on the decision for the plot directly to the south. However, it does not rely on the decision about the land directly north of it, because this decision is observed. None of the other decisions affect this agent's utility directly, and therefore they are not s-reachable.*

Intuitively, although the last pair of nodes in the relevance graph rely on each other, they rely on nothing else. Hence, we can compute an equilibrium for the pair together, regardless of any other decision rules. Once we have computed an equilibrium for this last pair, the decision variables can be treated as chance nodes, and we can proceed to compute an equilibrium for the next pair.

We formalize this intuition in the following definition:

Definition 9 *A set $\mathcal{S}$ of nodes in a directed graph is a* strongly connected component *(SCC) if for every pair of nodes $D \neq D' \in \mathcal{S}$, there exists a directed path from D to D'. A* maximal *SCC is an SCC that is not a strict subset of any other SCC.*

The maximal SCCs for the Road example are outlined in Fig. 4(b).

We can find the maximal SCCs of a relevance graph in linear time, by constructing a *component graph* whose nodes are the maximal SCCs of the graph [Cormen *et al.*, 1990]. There is an edge from component $\mathcal{C}$ to component $\mathcal{C}'$ in the component graph if and only if there is an edge in the relevance graph from some element of $\mathcal{C}$ to some element of $\mathcal{C}'$. The component graph is always acyclic, so we can define an ordering $\mathcal{C}_1, \ldots, \mathcal{C}_m$ over the SCCs, such that whenever $i < j$, no element of $\mathcal{C}_i$ is s-reachable from any element of $\mathcal{C}_j$.

We can now provide a divide-and-conquer algorithm for computing Nash equilibria in general MAIDs.

Algorithm 1

Given a MAID $\mathcal{M}$
a topological ordering $\mathcal{C}_1, \ldots, \mathcal{C}_m$ of the component graph derived from the relevance graph for $\mathcal{M}$
Let σ^0 be an arbitrary fully mixed strategy profile
For $i = 0$ through $m - 1$:
Let τ be a partial strategy profile for $\mathcal{C}_{(m-i)}$ that is a Nash equilibrium in $\mathcal{M}\left[\sigma^i_{-\mathcal{C}_{(m-i)}}\right]$
Let $\sigma^{i+1} = (\sigma^i_{-\mathcal{C}_{(m-i)}}, \tau)$
Output σ^m as an equilibrium of $\mathcal{M}$

The algorithm iterates backwards over the SCC's, finding an equilibrium strategy profile for each SCC in the MAID induced by the previously selected decision rules (with arbitrary decision rules for some decisions that are not relevant for this SCC). In this induced MAID, the only remaining decision nodes are those in the current SCC; all the other decision nodes have been converted to chance nodes. Finding the equilibrium in this induced MAID requires the use of a subroutine for finding equilibria in games. We simply convert the induced MAID into a game tree, as described in Section 3, and use a standard game-solving algorithm [McKelvey and McLennan, 1996] as a subroutine. Note that if the relevance graph is acyclic, each SCC consists of a single decision node. Thus, step 3 involves finding a Nash equilbrium in a single-player game, which reduces to simply finding a decision rule that maximizes the single agent's expected utility.

In proving the correctness of Algorithm 1, we encounter a subtle technical difficulty. The definition of strategic relevance (Def. 5) only deals with the optimality of a single decision rule for a strategy profile. But in Algorithm 1, we derive not just single decision rules, but a complete strategy for each agent. To make the leap from the optimality of single decision rules to the optimality of whole strategies in our proof, we must make the standard assumption of *perfect recall* — that agents never forget their previous actions or observations. More formally:

Definition 10 *An agent a has* perfect recall *with respect to a total order $D_1, \ldots, D_n$ over $\mathcal{D}_a$ if for all $D_i, D_j \in \mathcal{D}_a$, $i < j$ implies that $D_i \in Pa(D_j)$ and $Pa(D_i) \subset Pa(D_j)$.*

We can now prove the correctness of Algorithm 1.

Theorem 3 *Let $\mathcal{M}$ be a MAID where every agent has perfect recall, and let $\mathcal{C}_1, \ldots, \mathcal{C}_m$ be a topological ordering of the SCCs in the relevance graph for $\mathcal{M}$. Then the strategy profile σ^m produced by running Algorithm 1 with $\mathcal{M}$ and $\mathcal{C}_1, \ldots, \mathcal{C}_m$ as inputs is a Nash equilibrium for $\mathcal{M}$.*

To demonstrate the potential savings resulting from our algorithm, we tried it on the Road example, for different numbers of agents n. Note that the model we used differs slightly from that shown in Fig. 2: In our experiments, each agent had not just one utility node, but a separate utility node for each neighboring plot of land, and an additional node that depends on the suitability of the plot for different purposes. The agent's decision node is a parent of all these utility nodes. The idea is that an agent gets some base payoff for the building he builds, and then the neighboring plots and the suitability node apply additive bonuses and penalties to his payoff. Thus, instead of having one utility node with $3^5 = 243$ parent instantiations, we have 4 utility nodes with $3^2 = 9$ parent instantiations each. This change has no effect on the structure of the relevance graph, which is shown for $n = 6$ in Fig. 4(b). The SCCs in the relevance graph all have size 2; as we discussed, they correspond to pairs of decisions about plots that are across the road from each other.

Even for small values of n, it is infeasible to solve the Road example with standard game-solving algorithms. As we discussed, the game tree for the MAID has 3^{2n} leaves, whereas the MAID representation is linear in n. The normal form adds another exponential factor. Since each agent (except the first two) can observe three ternary variables, he has 27 information sets. Hence, the number of possible pure (deterministic) strategies for each agent is 3^{27}, and the number of pure strategy profiles for all n players is $(3^{27})^{(n-2)} \cdot (3^3)^2$. In the simplest interesting case, where $n = 4$, we obtain a game tree with 6561 terminal nodes, and standard solution algorithms, that very often use the normal form, would need to operate on a game matrix with about 4.7×10^{27} entries (one for each pure strategy profile).

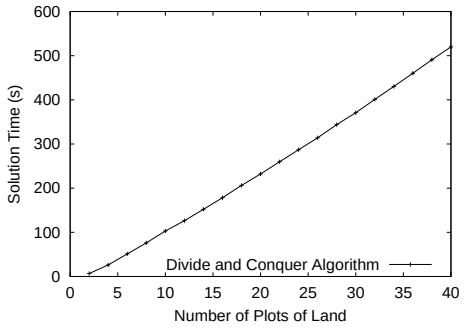


Figure 5: Performance results for the Road example.

Solving the Road game either in its extensive form or in the normal form is infeasible even for $n = 4$. By contrast, our divide-and-conquer algorithm ends up generating a sequence of small games, each with two decision variables. Fig. 5 shows the computational cost of the algorithm as n grows. We converted each of the induced MAIDs constructed during the algorithm into a small game tree, and used the game solver GAMBIT [2000] to solve it. As expected, the time required by our algorithm grows approximately linearly with n. Thus, for example, we can solve a Road MAID with 40 agents (corresponding to a game tree with 3^{80} terminal nodes) in 8 minutes 40 seconds.

6 Discussion and Future Work

We have introduced a new formalism, multi-agent influence diagrams (MAIDs), for modeling multi-agent scenarios with imperfect information. MAIDs use a representation where variables are the basic unit, and allow the dependencies between these variables to be represented explicitly, in a graphical form. They therefore reveal important qualitative structure in a game, which can be useful both for understanding the game and as the basis for algorithms that find equilibria efficiently. In particular, we have shown that our divide-and-conquer algorithm for finding equilibria provides exponential savings over existing solution algorithms in some cases, such as the Road example, where the maximal size of an SCC in the relevance graph is much smaller than the total number of decision variables. In the worst case, the relevance graph forms a single large SCC, and our algorithm simply solves the game in its entirety, with no computational benefits.

Although the possibility of extending influence diagrams to multi-agent scenarios was recognized at least fifteen years ago [Shachter, 1986], the idea seems to have been dormant

for some time. Suryadi and Gmytrasiewicz [1999] have used influence diagrams as a framework for learning in multi-agent systems. Milch and Koller [2000] use multi-agent influence diagrams as a representational framework for reasoning about agents' beliefs and decisions. However, the focus of both these papers is very different, and they do not consider the structural properties of the influence diagram representation, nor the computational benefits derived from it. Nilsson and Lauritzen [2000] have done related work on limited memory influence diagrams, but they focus on the task of speeding up inference in single-agent settings. MAIDs are also related to La Mura's [2000] game networks, which incorporate both probabilistic and utility independence. La Mura defines a notion of strategic independence, and also uses it to break up the game into separate components. However, his notion of strategic independence is an undirected one, and thus does not allow as fine-grained a decomposition as the directed relevance graph used in this paper, nor the use of a backward induction process for interacting decisions.

This work opens the door to a variety of possible extensions. On the representational front, it is important to extend MAIDs to deal with asymmetric situations, where the decisions to be made and the information available depend on previous decisions or chance moves. Game trees represent such asymmetry in a natural way, whereas in MAIDs (as in influence diagrams and BNs), a naive representation of an asymmetric situation leads to unnecessary blowup. We believe we can avoid these difficulties in MAIDs by explicitly representing context-specificity, as in [Boutilier *et al.*, 1996; Smith *et al.*, 1993], integrating the best of the game tree and MAID representations.

Another direction relates to additional structure that is revealed by the notion of strategic relevance. In particular, even if a group of nodes forms an SCC in the relevance graph, it might not be a fully connected subgraph; for example, we might have a situation where D_1 relies on D_2, which relies on D_3, which relies on D_1. Clearly, this type of structure tells us something about the interaction between the decisions in the game. An important open question is to analyze the meaning of these types of structures, and to see whether they can be exploited for computational gain. (See [Kearns *et al.*, 2001] for results in one class of MAIDs.)

Finally, the notion of strategic relevance is not the only type of insight that we can obtain from the MAID representation. We can use a similar type of path-based analysis in the MAID graph to determine which of the variables that an agent can observe before making a decision actually provide relevant information for that decision. In complex scenarios, especially those that are extended over time, agents tend to accumulate a great many observations. The amount of space needed to specify a decision rule for the current decision increases exponentially with the number of observed variables. Thus, there has been considerable work on identifying irrelevant parents of decision nodes in single-agent influence diagrams [Howard and Matheson, 1984; Shachter, 1990; 1998]. However, the multi-agent case raises subtleties that are absent in the single-agent case. This is another problem we plan to address in future work.

Acknowledgements This work was supported by Air Force contract F30602-00-2-0598 under DARPA's TASK program and by ONR MURI N00014-00-1-0637 under the program "Decision Making under Uncertainty".

References

[Boutilier *et al.*, 1996] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proc. 12th UAI*, pages 115–123, 1996.

[Cormen *et al.*, 1990] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[Fudenberg and Tirole, 1991] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

[Gambit, 2000] GAMBIT software, California Institute of Technology, 2000. http://www.hss.caltech.edu/gambit/Gambit.html.

[Geiger *et al.*, 1990] D. Geiger, T. Verma, and J. Pearl. Identifying independence in Bayesian networks. *Networks*, 20:507–534, 1990.

[Howard and Matheson, 1984] R. A. Howard and J. E. Matheson. Influence diagrams. In *Readings on the Principles and Applications of Decision Analysis*, pages 721–762. Strategic Decisions Group, 1984.

[Jensen *et al.*, 1994] F. Jensen, F.V. Jensen, and S.L. Dittmer. From influence diagrams to junction trees. In *Proc. 10th UAI*, pages 367–373, 1994.

[Kearns *et al.*, 2001] M. Kearns, M.L. Littman, and S. Singh. Graphical models for game theory. Submitted, 2001.

[Koller *et al.*, 1994] D. Koller, N. Megiddo, and B. von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Proc. 26th STOC*, pages 750–759, 1994.

[LaMura, 2000] P. LaMura. Game networks. In *Proc. 16th UAI*, pages 335–342, 2000.

[McKelvey and McLennan, 1996] R.D. McKelvey and A. McLennan. Computation of equilibria in finite games. In *Handbook of Computational Economics*, volume 1, pages 87–142. Elsevier Science, Amsterdam, 1996.

[Milch and Koller, 2000] B. Milch and D. Koller. Probabilistic models for agents' beliefs and decisions. In *Proc. 16th UAI*, 2000.

[Nash, 1950] J. Nash. Equilibrium points in n-person games. *Proc. National Academy of Sciences of the USA*, 36:48–49, 1950.

[Nilsson and Lauritzen, 2000] D. Nilsson and S.L. Lauritzen. Evaluating influence diagrams with LIMIDs. In *Proc. 16th UAI*, pages 436–445, 2000.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Francisco, 1988.

[Romanovskii, 1962] I. V. Romanovskii. Reduction of a game with complete memory to a matrix game. *Soviet Mathematics*, 3:678–681, 1962.

[Shachter, 1986] R. D. Shachter. Evaluating influence diagrams. *Operations Research*, 34:871–882, 1986.

[Shachter, 1990] R. D. Shachter. An ordered examination of influence diagrams. *Networks*, 20:535–563, 1990.

[Shachter, 1998] R. D. Shachter. Bayes-ball: The rational pastime. In *Proc. 14th UAI*, pages 480–487, 1998.

[Smith *et al.*, 1993] J. E. Smith, S. Holtzman, and J. E. Matheson. Structuring conditional relationships in influence diagrams. *Operations Research*, 41(2):280–297, 1993.

[Suryadi and Gmytrasiewicz, 1999] D. Suryadi and P.J. Gmytrasiewicz. Learning models of other agents using influence diagrams. In *Proc. 7th Int'l Conf. on User Modeling (UM-99)*, pages 223–232, 1999.

[Zermelo, 1913] E. Zermelo. Über eine Anwendung der Mengenlehre auf der Theorie des Schachspiels. In *Proceedings of the Fifth International Congress on Mathematics*, 1913.

MULTI-AGENT SYSTEMS

MULTI-AGENT SYSTEMS

Multiagent Coordination by Stochastic Cellular Automata

T D Barfoot and **G M T D'Eleuterio**
tim.barfoot@utoronto.ca, gde@utias.utoronto.ca
University of Toronto Institute for Aerospace Studies
4925 Dufferin Street, Toronto, Ontario, Canada, M3H 5T6

Abstract

A coordination mechanism for a system of sparsely communicating agents is described. The mechanism is based on a stochastic version of cellular automata. A parameter similar to a temperature can be tuned to change the behaviour of the system. It is found that the best coordination occurs near a phase transition between order and chaos. Coordination does not rely on any particular structure of the connections between agents, thus it may be applicable to a large array of sparsely communicating mobile robots.

1 Introduction

The term *multiagent system* encompasses large bodies of work from engineering, computer science, and mathematics. Examples include networks of mobile robots [Matarić, 1992], software agents [Bonabeau *et al.*, 1994], and cellular automata [Wolfram, 1984]. A common thread in all multiagent systems is the issue of *coordination*. How are a large number of sparsely coupled agents able to produce a coherent global behaviour using simple rules? Answering this question will not only permit the construction of interesting and useful artificial systems but may allow us to understand more about the natural world. Ants and the other social insects are perfect examples of local interaction producing a coherent global behaviour. It is possible for millions of ants to act as a *superorganism* through local pheromone communication. We seek to reproduce this ability on a fundamental level in order to coordinate artificial systems.

It can be argued that *cellular automata* (CA) are the simplest example of a multiagent system. Originally studied by [von Neumann, 1966], the term CA is used to describe systems of sparsely coupled difference equations. Despite their simple mechanics, some extremely interesting behaviours have been catalogued (e.g., Conway's *Game of Life*). The word *self-organization* is used in many contexts when discussing multiagent systems which can lead to confusion. Here we use it to mean *multiagent coordination in the face of more than one alternative*. We will be describing a stochastic version of cellular automata. The goal will be to have all cells choose the same symbol from a number of possibilities using only sparse communication. We maintain that rules able to succeed at this task are self-organizing because the cells are not told which symbol to choose, yet they must all coordinate their choices to produce a globally coherent decision. If we told the cells which symbol to choose, the task would be very easy and no communication between cells would be necessary. This can be dubbed *centralized organization* and is in stark contrast to *self- or decentralized organization*. We believe that coordination in the face of more than one alternative is at the very heart of all multiagent systems.

This paper is organized as follows. Related work is described, followed by a description of the model under consideration. Results of its performance on the multiagent coordination task are presented. Statistical analysis of the rule are provided followed by discussions and conclusions.

2 Related Work

In the following note that typically cellular automata do not operate in a stochastic but rather a deterministic manner. Unless explicitly stated (e.g., *stochastic cellular automata* (SCA)), the term cellular automata will imply determinism.

[von Neumann, 1966] originally studied cellular automata in the context of self-reproducing mechanisms. The goal was to devise local rules which would reproduce and thus spread an initial pattern over a large area of cells, in a tiled fashion. The current work can be thought of as a simple case of this where the tile size is only a single cell but there are multiple possibilities for that tile. Futhermore, we wish our rules to work starting from any random initial condition of the system.

Cellular automata were categorized by the work of [Wolfram, 1984] in which four *universality classes* were identified. All rules were shown to belong to one of class I (fixed point), class II (oscillatory), class III (chaotic), or class IV (long transient). These universality classes can also be identified in SCA and we will show that in our particular model, choosing a parameter such that the system displays long transient behaviour (e.g., class IV) results in the best performance on our multiagent coordination task.

[Langton, 1990] has argued that natural computation may be linked to the universality classes. It was shown that by tuning a parameter to produce different CA rules, a phase transition was exhibited. The relation between the phase transition and the universality classes was explored. It was found that class IV behaviour appeared in the vicinity of the phase transition. The current work is very comparable to this study in

that we also have a parameter which can be tuned to produce different CA rules. However, our parameter tunes the amount of randomness that is incorporated into the system. At one end of the spectrum, completely random behaviour ensues while at the other completely deterministic behaviour ensues. We also relate the universality classes to particular ranges of our parameter and find a correlation between performance on our multiagent coordination task and class IV behaviour. We attempt to use similar statistical measures to [Langton, 1990] to quantify our findings.

[Mitchell *et al.*, 1993], [Das *et al.*, 1995] study the same coordination task as will be examined here in the case of deterministic CA. However, their approach is to use a genetic algorithm to evolve rules successful at the task whereas here hand-coded rules are described. They found that the best solutions were able to send long range *particles* (similar to those in the *Game of Life*) [Andre *et al.*, 1997] in order to achieve coordination. These particles rely on the underlying structure of the connections between cells, specifically that each cell is connected to its neighbours in an identical manner. The current work assumes that no such underlying structure may be exploited and that the same mechanism should work for different connective architectures. The cost for this increased versatility is that the resulting rules are less efficient (in terms of time to coordinate) than their particle-based counterparts.

[Tanaka-Yamawaki *et al.*, 1996] studies the same problem to that considered here. They use *totalistic* [Wolfram, 1984] rules which do not permit exploitation of the underlying structure of the connections between cells but rather rely on the intensity of each incoming symbol. They also vary a parameter to produce different rules and find that above a certain threshold, "global consensus" occurs but below it does not. However, they consider large clusters of symbols to be a successful global consensus. We do not and thus turn to a stochastic version of their totalistic rules to destroy these clusters and complete the job of global coordination.

3 The Model

In deterministic cellular automata there is an *alphabet* of K symbols, one of which may be adopted by each cell. Incoming connections each provide a cell with one of these symbols. The combination of all incoming symbols uniquely determines which symbol the cell will display as output. Stochastic cellular automata (SCA) work in the very same way except at the output level. Instead of there being a single unique symbol which is adopted with probability 1, there can be multiple symbols adopted with probability less than 1. Based on this *outgoing probability distribution* over the K symbols, a single unique symbol is drawn to be the output of the cell. This is done for all cells simultaneously. It should be noted that deterministic CA are a special case of SCA.

We consider a specific sub-case of SCA in this paper which corresponds to the totalistic rules of CA. Assume that cells cannot tell which symbols came from which connections. In this case, it is only the intensity of each incoming symbol which becomes important. Furthermore, we desire that our rules work with any number of incoming connections thus rather than using the number of each of the incoming K symbols, we use this number normalized by the number of connections which can be thought of as an *incoming probability distribution*. In summary the model we consider is as follows.

Totalistic SCA. *Consider a system of N cells, each of which is connected to a number of other cells. Let $\mathcal{A}$ represent an alphabet of K symbols. The* state *of Cell i at time-step t is $x_i[t] \in \mathcal{A}$. The* input probability distribution, $\boldsymbol{p}_{\text{in}}$, *for Cell i is given by*

$$\boldsymbol{p}_{\text{in}}[t] = \boldsymbol{\sigma}_i\big(\, x_1[t], \ldots, x_N[t]\,\big) \tag{1}$$

where $\boldsymbol{\sigma}_i$ accounts for the connections of Cell i to the other cells. The output probability distribution $\boldsymbol{p}_{\text{out}}$ *is given by the map, φ,*

$$\boldsymbol{p}_{\text{out}}[t+1] = \varphi\big(\boldsymbol{p}_{\text{in}}[t]\big) \tag{2}$$

The probability distributions $\boldsymbol{p}_{\text{in}}$ and $\boldsymbol{p}_{\text{out}}$ are stochastic columns. *The new state of Cell i at time-step $t+1$ is randomly drawn according to the distribution $\boldsymbol{p}_{\text{out}}[t+1]$ and is represented by $x_i[t+1]$.*

It should be noted that in (1) if the connections between the cells are not changing over time then the functions, $\boldsymbol{\sigma}_i(\cdot)$, will not be functions of time. However, we could allow these connections to change which would make them functions of time.

Once the connections are described through the $\boldsymbol{\sigma}_i(\cdot)$ functions, the only thing that remains to be defined is the φ-map. We assume that each cell has the same φ-map but this need not be the case. The possibilities for this map are infinite and thus for the remainder of this paper we discuss a parameterized subset of these possibilities. This subset will be called *piecewise-φ* and is defined as follows.

Piecewise-φ. *Let*

$$\boldsymbol{p}_{\text{in}} = \big[\, p_{1,\text{in}} \;\cdots\; p_{K,\text{in}} \,\big]^T \tag{3}$$

The (unnormalized) output probabilities are given by

$$p_{k,\text{out}} = \begin{cases} 1, & \text{if } \frac{1}{K} + \beta(p_{k,\text{in}} - \frac{1}{K}) \geq 1 \\ 0, & \text{if } \frac{1}{K} + \beta(p_{k,\text{in}} - \frac{1}{K}) \leq 0 \\ \frac{1}{K} + \beta(p_{k,\text{in}} - \frac{1}{K}), & \text{otherwise} \end{cases} \tag{4}$$

where β is derived from the tunable parameter λ as follows:

$$\beta = \begin{cases} 2\lambda, & \text{if } 0 \leq \lambda \leq \frac{1}{2} \\ \frac{1}{2}(1-\lambda)^{-1}, & \text{if } \frac{1}{2} \leq \lambda < 1 \end{cases} \tag{5}$$

The (normalized) output probability column is

$$\boldsymbol{p}_{\text{out}} = \frac{1}{p_{\Sigma,\text{out}}} \big[\, p_{1,\text{out}} \;\cdots\; p_{K,\text{out}} \,\big]^T \tag{6}$$

where $p_{\Sigma,\text{out}} = \sum_{k=1}^{K} p_{k,\text{out}}$.

Note that in (5), the tunable parameter, λ acts in a similar manner to a temperature parameter. When $\lambda = 1$ we have a completely deterministic rule while when $\lambda = 0$ we have a completely random rule. Figure 1 shows what the rule looks like for different λ when $K = 2$.

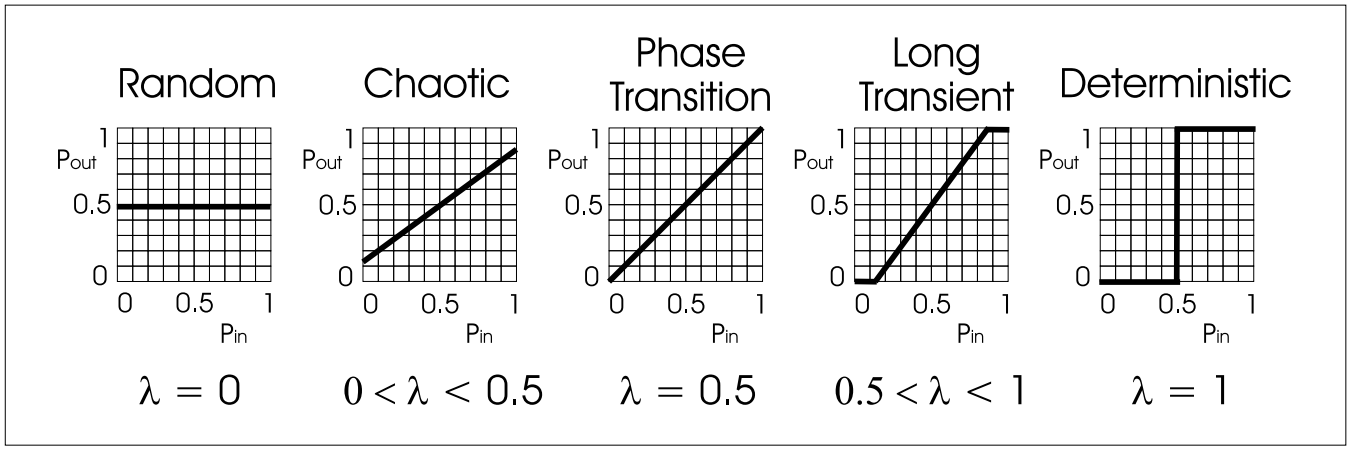


Figure 1: The *piecewise-*φ rule for different values of λ and $K = 2$.

An equilibrium point, $\boldsymbol{p}^*$, in a φ-map is one for which the following is true

$$\boldsymbol{p}^* = \varphi(\boldsymbol{p}^*) \tag{7}$$

The idea behind the *piecewise-*φ rule was to create an instability in the probability map at the uniform distribution equilibrium point

$$\boldsymbol{p}^*_{\text{uni}} = \left[\begin{array}{cccc} \frac{1}{K} & \frac{1}{K} & \cdots & \frac{1}{K} \end{array} \right]^T \tag{8}$$

such that a small perturbation from this point would drive the probability towards one of the stable equilibria

$$\boldsymbol{p}^*_1 = [\, 1 \;\; 0 \;\; \cdots \;\; 0 \,]^T \tag{9}$$

$$\boldsymbol{p}^*_2 = [\, 0 \;\; 1 \;\; \cdots \;\; 0 \,]^T \tag{10}$$

$$\vdots \tag{11}$$

$$\boldsymbol{p}^*_K = [\, 0 \;\; 0 \;\; \cdots \;\; 1 \,]^T \tag{12}$$

It turns out that when $0 \leq \lambda < \frac{1}{2}$, the equilibrium point, $\boldsymbol{p}^*_{\text{uni}}$, is the only stable equilibrium. However when $\frac{1}{2} < \lambda \leq 1$, $\boldsymbol{p}^*_{\text{uni}}$ becomes unstable and the other equilibria, $\boldsymbol{p}^*_1, \ldots, \boldsymbol{p}^*_K$, become stable. This is similar to the classic pitchfork bifurcation as depicted in figure 2 for $K = 2$. However, with K symbols in the alphabet the pitchfork will have K tines.

It is important to stress that we have designed the stability of our system at a local level. The question of global stability and success on the multiagent coordination problem does not follow directly from the local stability of each cell. It might be possible to study the global stability of a system of cells with the *piecewise-*φ rule analytically. The approach in this paper has been to study it through simulation and statistical measures.

4 Simulation

We now present simulations of cells running the *piecewise-*φ rule. In order to ensure that the connections between cells are not regular, we consider each cell to exist in a Cartesian box (of size 1 by 1). The N cells are randomly positioned in this box and symmetrical connections are formed between two cells if they are closer than a threshold Euclidean distance, d, from one another. Figure 4 shows example connections between $N = 100$ cells with $d = 0.2$. Figure 3 shows example time series for different values of λ. When $\lambda < 0.5$, chaotic global behaviour arises, with $0.5 < \lambda < 1$ fairly successful behaviour results but with $\lambda = 1$ clusters form. The formation of clusters means that the global system has stable equilibria which we did not predict from the local rule. However, as λ is decreased towards 0.5, these equilibria are no longer stable and the system continues to coordinate.

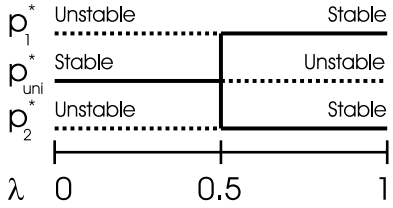


Figure 2: Pitchfork stability of the *piecewise-*φ rule for $K = 2$. λ is a parameter analogous to a temperature.

It would seem that there is a good correlation between the stability on the local level and the behaviour type of the global system. As λ moves from below 0.5 to above, it appears there is a dramatic phase transition in the behaviour of the system (totally chaotic to fixed point). In the neighbourhood of 0.5 there is long transient behaviour. It turns out that the best value for λ, from the point of view of multiagent coordination, is approximately $\lambda = 0.6$.

5 Statistics

In an attempt to quantify the qualitative observations of the previous section a number of statistical measures were employed in the analysis of the SCA time series. These were used also by [Langton, 1990]. The first measure is taken from [Shannon, 1948] and will be referred to as *entropy* (H). It is defined as follows.

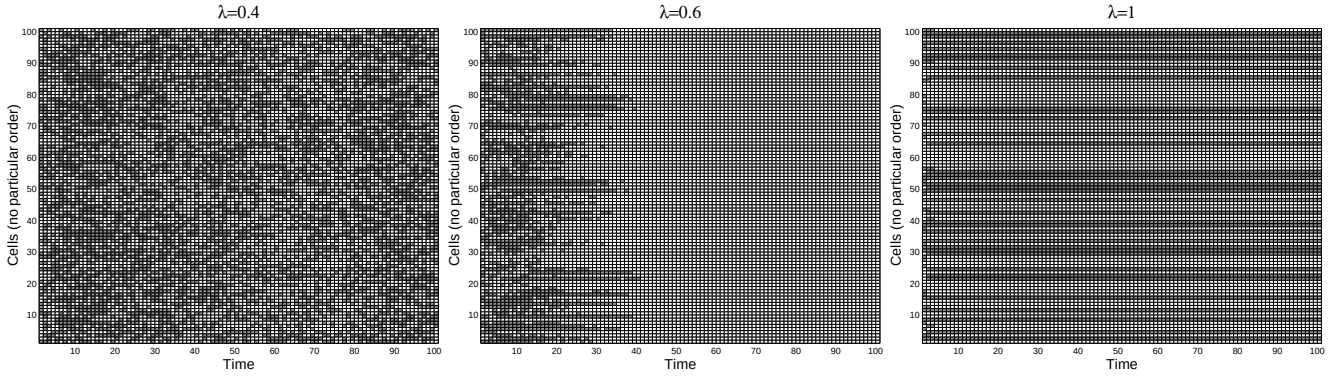


Figure 3: Example time series for different values of λ and $N = 100$, $K = 2$, $d = 0.2$. (left) chaotic behaviour, (middle) successful coordination, (right) clusters. The 2 colours represent the 2 symbols of the alphabet.

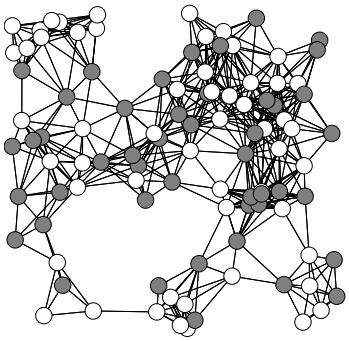


Figure 4: Example connections between $N = 100$ cells with $d = 0.2$. K colours represent the K symbols of the alphabet. Here an initial random condition is displayed for $K = 2$.

Entropy *Given a sequence of M symbols*

$$\mathbf{s} = [\ s_1 \quad s_2 \quad \cdots \quad s_M\ ]^T \tag{13}$$

from an alphabet of size K, the entropy *of the sequence may be computed as follows. First compute the frequency, n_k, of each of the K symbols $\forall k \in 1 \ldots K$ which is simply the number of occurrences of symbol k in the sequence, $\mathbf{s}$. From the frequencies, compute the probability, p_k, of each of the K symbols $\forall k \in 1 \ldots K$ as*

$$p_k = \frac{n_k}{n_\Sigma} \tag{14}$$

where $n_\Sigma = \sum_{i=1}^{K} n_i$. Finally, the entropy of sequence, $H(\mathbf{s})$, is defined as

$$H(\mathbf{s}) = \frac{-\sum_{k=1}^{K} p_k \ln(p_k)}{\ln(K)} \tag{15}$$

where the $\ln(K)$ denominator is a normalization constant to make $H(\mathbf{s}) \in [0, 1]$.

This entropy function produces a value of 0 when all the symbols in $\mathbf{s}$ are identical and a value of 1 when all K symbols are equally common. The second measure is based on the first and will be referred to as *mutual information* (I). It is defined as

Mutual Information *Given two sequences of M symbols*

$$\mathbf{s}_1 = [\ s_{1,1} \quad s_{1,2} \quad \cdots \quad s_{1,M}\ ]^T \tag{16}$$

$$\mathbf{s}_2 = [\ s_{2,1} \quad s_{2,2} \quad \cdots \quad s_{2,M}\ ]^T \tag{17}$$

from an alphabet of size K, the mutual information *of the sequence, $I(\mathbf{s}_1, \mathbf{s}_2)$, may be defined as*

$$I(\mathbf{s}_1, \mathbf{s}_2) = H(\mathbf{s}_1) + H(\mathbf{s}_2) - H(\mathbf{s}_1, \mathbf{s}_2) \tag{18}$$

where $H(\mathbf{s}_1, \mathbf{s}_2)$ is the entropy of the two sequences considered as a joint process (i.e., with an alphabet of size $K \times K$).

These two measures may be computed on any sequence of symbols. We tested them on *spatial* sequences (e.g., time series *columns* from figure 3) and *temporal* sequences (e.g., time series *rows* from figure 3). The most interesting measures were *average spatial entropy* (average of entropies computed from all columns in a time series) and *average temporal mutual information* (average of all Is computed from all rows in a time series. I was computed between a row and itself shifted by one time-step).

Figure 5 show various measures for 1000 values of λ. At each value of λ, 100 simulations were done on different random connections between cells and initial conditions. Thus, all displayed measures are actually averaged over 100 simulations. Each simulation was run for 300 time-steps with $N = 100$, $K = 2$, and $d = 0.2$.

Figure 5 (left) shows the average number of clusters[1] at the final time-step for different values of λ. Clearly there is an optimal value of λ near 0.6. Figure 5 (middle) shows average spatial entropy for different values of λ. This measure has a good correlation with average number of clusters. Again, there is a minimum occurring at approximately $\lambda = 0.6$ which corresponds to the best performance at multiagent coordination.

Figure 5 (right) displays average temporal mutual information for different values of λ. This is a very interesting plot.

[1]Number of clusters was computed by considering the SCA as a Markov chain with connections deleted between cells displaying different symbols. The number of clusters is then the number of eigenvalues equal to 1 from the Markov transition matrix.

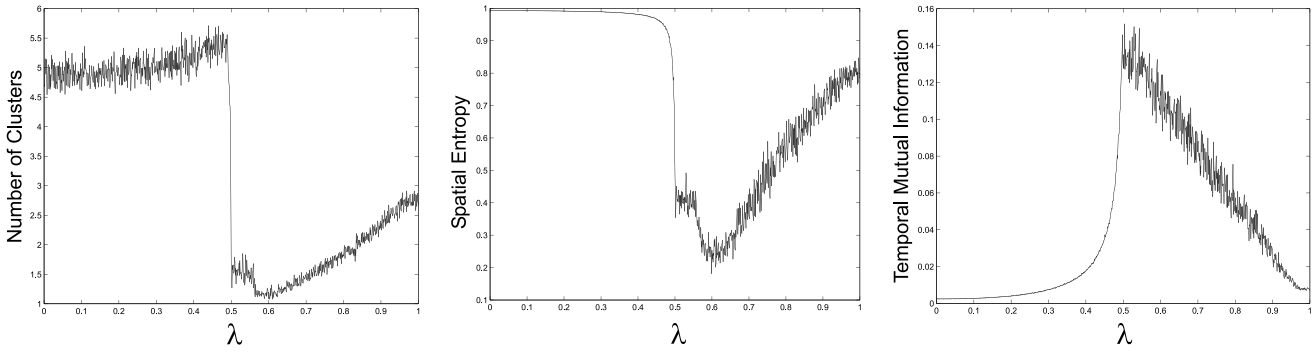


Figure 5: (left) Average number of clusters at final time-step for 1000 values of λ. (middle) Average spatial entropy for 1000 values of λ. (right) Average temporal mutual information for 1000 values of λ. All plots show average of 100 simulations at each value of λ.

Temporal mutual information seems to capture the length of the global transient behaviour of the system. As discussed in [Langton, 1990], the random pattern in the chaotic region is not considered transient but rather the steady state behaviour. The peak in temporal mutual information occurs at $\lambda = 0.5$, the phase transition, and drops away on either side. [Langton, 1990] has a similar plot. Figure 6 shows how the number of clusters at the final time-step (300 time-steps) changes as the problem scales up to more cells; it appears to be a linear relationship. Figure 7 shows how the number of clusters at the final time-step changes for different message sizes.

6 Discussion

The strong correlation between the local stability of the *piecewise-φ* rule and the type of global behaviour is quite interesting. It appears that $\lambda \gg 0.5$ corresponds to fixed point behaviour (class I), $\lambda \ll 0.5$ corresponds to chaotic behaviour (class III), and λ near 0.5 corresponds to long transient behaviour (class IV). The correlation most likely has something to do with the way in which the incoming probability distribution is computed in (1). This step delivers information averaged from all connected cells. This averaging serves to smooth out differences between connected cells. However, if this smoothing occurs too quickly (i.e., $\lambda = 1$) the system does not have time to smooth globally resulting in the formation of clusters. The addition of noise in the particular form of the *piecewise-φ* rule aids in slowing the smoothing process thus destroying the clusters. This has been called *critical slowing down* [Haken, 1983] in other systems. As we approach the critical point ($\lambda = 0.5$ or $\beta = 1$) from above, the strength of the instability decreases which slows down the decision-making process. It is a balance of these two effects which seems to be the most effective at multiagent coordination. The optimal operating value of λ is not right at the phase transition but a little bit towards the deterministic end of the λ spectrum (approximately $\lambda = 0.6$).

Note that we did not find any oscillatory behaviour (class II) which is likely because the connections between the cells are symmetrical. However, if the *piecewise-φ* rule in figure 1 is reflected (left-right) then the system 'blinks' and global coordination corresponds to all cells blinking in phase with one another.

In this model of multiagent coordination, the boundaries between clusters have purposely been made unstable. This forces them to move randomly until they contact one another and annihilate, leaving a single cluster. The results presented here used $N = 100$ cells and required on average 150 time-steps to get to a single cluster with $d = 0.2$, $K = 2$ and $\lambda = 0.6$. Clearly the time required to form a single cluster will increase with the number of cells in the system. Figure 6 confirms this by showing that at the end of 300 time-steps, increasing the number of cells, N, results in more clusters. The linear relationship suggests that scaling-up may be possible but more in depth studies are required. Figure 7 shows how the system scales to different message sizes, K. Here the relationship between number of clusters (after 300 time-steps) to message size is a bit surprising, first dropping then increasing as K increases. It levels off again as the number of symbols exceeds the number of cells (since at most N symbols can be represented in the random initial condition). Again, the nature of this scaling should be studied more closely.

The *piecewise-φ* rule is not the only map that can be used to achieve multiagent coordination in SCA. Replacing it with other monotonically increasing functions (i.e., in figure 1) with the same equilibria also works. We had comparable success to the piecewise-φ map using

$$p_{k,\text{out}} = (p_{k,\text{in}})^{\beta} \quad \forall k \in 1 \ldots K \tag{19}$$

with the outgoing probability column normalized as in (6).

The model considered here does not require knowledge of the underlying structure of the connections between cells. This was a design requirement as it was originally motivated by a network of communicating mobile robots whose connections might be changing over time and thus difficult to exploit. It is thus natural to question whether the model still works as the connections are varied over time. To this end, a small amount of Gaussian noise was added to the positions of the cells in the Cartesian box of figure 4 at each time-step. As the cells moved, the connections between them changed (since they are limited by the range, d). The SCA model was still able to form single clusters. This was possible even when $\lambda = 1$ which does make sense since there is still some noise being added. However, the nature of the noise is at the connection level rather than the signal level. This aspect is currently under further investigation.

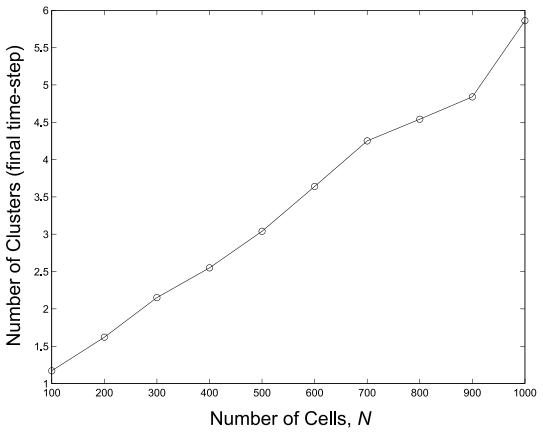


Figure 6: Number of clusters at final time-step as the number of cells is varied from $N = 100$ to $N = 1000$. Parameters for each run were 300 time-steps, $\lambda = 0.6$, $K = 0.2$ and $d = \frac{0.2}{\sqrt{N}}$ to keep the density of connections constant. Plot shows average of 100 simulations at each value of N.

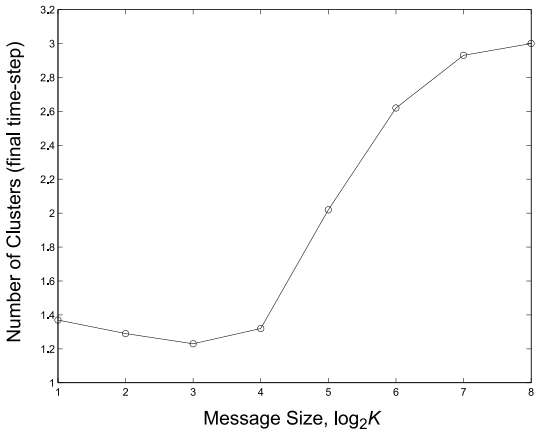


Figure 7: Number of clusters at the final time-step as the message size is varied from $K = 2$ (1 bit) to $K = 256$ (8 bits). Parameters for each run were 300 time-steps, $\lambda = 0.6$, $N = 100$ and $d = 0.2$. Plot shows average of 100 simulations at each value of K.

7 Conclusion

A mechanism for multiagent coordination has been presented based on stochastic cellular automata. We consider this to be an example of self-organizing behaviour in that global coordination occurs in the face of more than one alternative. It was shown that by using stochastic rules, sparsely communicating agents could come to a global consensus. A parameter in the coordination mechanism was tuned and it was found that coordination occurred best when the system was near a phase transition between chaotic and ordered behaviour (the optimum was a little bit towards the ordered side).

It is hoped that this model will shed light on self-organization as a general concept while at the same time providing a simple algorithm to be used in practice.

Acknowledgements

We would like to acknowledge support of this work by NSERC (Natural Science and Engineering Research Council) and CSA (Canadian Space Agency).

References

[Andre *et al.*, 1997] David Andre, Forrest H. Bennett, and John R. Koza. Evolution of intricate long-distance communication signals in cellular automata using genetic programming. In Chris G. Langton and Katsunori Shimohara, editors, *Artificial Life V, Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*. MIT Press, 1997.

[Bonabeau *et al.*, 1994] Eric Bonabeau, Guy Theraulaz, Eric Arpin, and Emmanual Sardet. The building behaviour of lattice swarms. In Rodney A. Brooks and Pattie Maes, editors, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*. MIT Press, 1994.

[Das *et al.*, 1995] Rajarshi Das, James P. Crutchfield, Melanie Mitchell, and James E. Hanson. Evolving globally synchronized cellular automata. In L.J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 336–343, San Fransisco, CA, April 1995.

[Haken, 1983] H. Haken. *Synergetics, An Introduction, 3rd Edition*. Springer-Verlag, Berlin, 1983.

[Langton, 1990] Chris G. Langton. Computations at the edge of chaos: Phase transitions and emergent computation. *Physica D*, 42:12–37, 1990.

[Matarić, 1992] Maja J. Matarić. Designing emergent behaviours: From local interactions to collective intelligence. In J. A. Meyer, H.L. Roitblat, and S. W. Wilson, editors, *Simulation of Adaptive Behaviour: from Animals to Animats 2*. MIT Press, 1992.

[Mitchell *et al.*, 1993] Melanie Mitchell, Peter T. Hraber, and James P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993. SFI Working Paper 93-03-014.

[Shannon, 1948] C E Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, July 1948.

[Tanaka-Yamawaki *et al.*, 1996] Mieko Tanaka-Yamawaki, Sachiko Kitamikado, and Toshio Fukuda. Consensus formation and the cellular automata. *Robotics and Autonomous Systems*, 19:15–22, 1996.

[von Neumann, 1966] Jon von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana and London, 1966.

[Wolfram, 1984] Stephen Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

Identifying the Scope of Modeling for Time-Critical Multiagent Decision-Making*

Sanguk Noh
Computer Science Department
University of Missouri-Rolla
nohs@umr.edu

Piotr J. Gmytrasiewicz
Dept. of Computer Science and Engineering
University of Texas at Arlington
piotr@cse.uta.edu

Abstract

Decision-making in multiagent settings requires significant computational resources. Agents need to model each other to decide how to coordinate – this sometimes may require solving nested models of many other agents and may be impractical to perform in an acceptable time. In this paper, we investigate ways in which the agents can be equipped with flexible decision-making procedures to allow multiagent decision-making under time pressure. One of the techniques we implemented uses iterative deepening algorithm guided by performance profiles generated offline. When the interaction involves many agents, the algorithm iteratively enhances the quality of coordinated decision-making by incrementally adding the levels of nesting considered, but with the additional penalty of increased running time. To identify the appropriate scope of modeling online we use the concept of urgency which represents cost of delaying decisions. We validate our framework with experiments in a simulated anti-air defense domain. The contribution of our framework is that it endows our autonomous agents with flexibility to cope with time pressure in complex multiagent settings.

1 Introduction

In multiagent settings, deliberative decision-theoretic methods [Gmytrasiewicz *et al.*, 1998; Gmytrasiewicz and Durfee, 2000; Noh and Gmytrasiewicz, 1999a; 1999b; Vidal and Durfee, 1995] allow an agent to compute its best action, i.e., with the highest expected utility, given its available knowledge. The agent's knowledge may contain information about environment, about the agent itself, about other agents present, about what the other agents know, and so on. Therefore, when making coordinated decisions in complex scenarios with many agents present, the agent may have to consider large and complex knowledge structures, which may lead to it failing to decide an action in an acceptable time.

In response to these issues we investigate methods which allow the agents to perform multiagent decision-making in time-critical situations. In our earlier work [Noh and Gmytrasiewicz, 1999b] we concentrated on compiling results of deliberative decision-making using machine learning and creating reactive rules that improved the agent's reactivity. This current paper focuses on limiting deliberation time by identifying the proper scope of knowledge used for decision-making. We wish to establish which information, assuming it is in the agent's possession in the first place, can be neglected and at what cost. In general, there are two approaches which we call simplification and approximation techniques:

- *Simplifications* involve neglecting a certain part of available information and computation associated with processing this information, while preserving the accuracy of deliberation. In general, there is no guarantee that a simplification technique will be applicable in any particular situation but, if it is, it can be used without sacrificing the quality of the result.
- *Approximations* involve limiting information considered while making decisions without the guarantee that the solution obtained is optimal.

In our framework, the simplification techniques remove computations that are found not to influence the quality of output. For example, if the agent finds that one of its actions is better than all other actions no matter what (we call it the dominant action) then any further computations of the values of actions become unnecessary. Other examples include memoization, and pruning certain parts of structured knowledge representation (defined rigorously in [Gmytrasiewicz and Durfee, 2000] and presented in Figure 1), if it can be proven that they do not influence the solution. All simplifications reduce the computation time needed for deliberation without significant overhead. The approximation techniques determine what computational resources should be used for agent's decision making. These methods consider a tradeoff between decision quality and computation time.

*This research has been sponsored by the Office of Naval Research Artificial Intelligence Program under contract N00014-95-1-0775 and by the National Science Foundation CAREER award IRI-9702132.

In other words, the objective of approximation techniques is to provide autonomous agents with the boundedly rational decision, given the real-time demands of the situation at hand. The approximation techniques include: (1) using compiled rules to constrain alternative actions, on which we reported in [Noh and Gmytrasiewicz, 1999b], and (2) using performance profiles and urgency of situation to determine the appropriate scope of modeling.

In this paper, we will explore the second of the approximation techniques. We will identify the appropriate scope of modeling of other agents based on performance profiles and the calculation of urgency. Toward this end, we focus on the amount of information that can be included in modeling other agents. We are inspired by techniques used by humans in real-life situations when they make coordinated decisions with a large number of individuals. Frequently, we consider and think about only a few agents in our immediate vicinity for the purpose of coordination. Similarly, our agents start by considering their closest neighbors first, and then broaden the scope by including more distant agents (in Figure 1 this would correspond to broadening of, what we call the modeling structure.) At the same time, our agents need to decide on how deep the modeling should proceed. Should one consider how the other agents model others, for example? While reasoning at deeper levels results in better decisions, it also requires more computation time.

Thus, the entire modeling space has the dimension of the *breadth*, which corresponds to the number of agents considered for coordination, and the *depth*, which corresponds to the reasoning level. Based on the structure of the modeling space, we present an iterative deepening algorithm which provides our agents with preemptible results according to different breadths and depths of the modeling space. This algorithm can be used offline to generate *performance profiles* [Dean, 1990; Zilberstein and Russell, 1996], which illustrate the performance vs. computation time required by processing increasingly more information. Further, we introduce the concept of urgency which represents the value of time given a situation at hand. For example, if a situation is very urgent, the agent should be ready to execute an action quickly, without much reasoning efforts. When the situation is not urgent, on the other hand, the agent can afford to consider more information and spend more deliberation time looking for better decisions. Our agents decide whether to act or think by computing urgency.

2 Related Work

We explore the ways to reduce the complexity of deliberative decision-making procedures that might cause autonomous agents to fail in accomplishing their tasks under time pressure, and, at the same time, to preserve the accuracy of deliberation as much as possible. Due to the impracticality of an agent with perfect rationality, it has been proposed that bounded optimality can be a way to yield a solution within a specific time in spite of sacrificing the optimality of the decision to some degree. Thus, our work builds on efforts by several other researchers who focus on making autonomous systems tractable when there is no sufficient time to produce optimal solutions in complex environments.

One approach uses a purely reactive reasoning procedure. The goal of reactive systems is to directly respond to the condition with a predefined action. Although such a system is able to react very quickly to environments, plans will be useless if problem varies and re-planning is needed. Further, if no protocol or overall plan can be established explicitly in advance, this reactive system is unrealistic or likely to lock the agents into suboptimal forms of behavior. However, our deliberative reasoning procedure provides rational behaviors in dynamic environments even when plans are not pre-defined.

Other rigorous efforts have focused on bounding the rationality of deliberative agents, which limits the resources available to agents. These approaches include work on the use of meta-reasoning procedures to control inference [Horvitz, 1988; Russell and Wefald, 1991], and anytime algorithms [Dean, 1990; Zilberstein and Russell, 1996]. Meta-reasoning and anytime algorithms balance the costs and the benefits of continued deliberation [Good, 1971]. They perform online computation of expected utilities of additional deliberation. In contrast, our approach incorporates offline empirical data (performance profiles) into the decision of whether to act or think. By using the information calculated offline, our meta-reasoning is faster than their meta-reasoning.

In the field of multiagent coordination, there have been several approaches to support practical decision-making. Lesser [Lesser, 1998] proposes a domain independent agent architecture for a large-scale multiagent environment, which deals with adaptation to resource availability, tradeoffs between resources and quality, complex interdependencies among agents, and so forth. For further research, the components of their software infrastructure should be implemented and tested in sophisticated, real-time multiagent applications. Durfee [Durfee, 1999] suggests strategies for making coordination practical when agents use the Recursive Modeling Method (RMM) [Gmytrasiewicz and Durfee, 2000] as a nested model framework. These strategies include simplifying utility computations, limiting possible choices, treating multiple agents as one, and so on. This work is similar to our current work in that both approaches use RMM to make agents' rational decisions. However, we identify the scope of modeling knowledge processed by an agent considering the urgency of the situation at hand whereas Durfee simplifies (or ignores) computational resources as much as possible. Further, our work concentrates on offline efforts to make real-time decision tractable but most of Durfee's strategies for practical coordination have to be executed online. Lastly, in our own work [Noh and Gmytrasiewicz, 1999b], we propose an adaptive and deliberative agent architecture consisting of compiled and deliberative decision procedures. We compile the results of deliberative decision-making into a set of reactive

condition-action rules with numerous machine learning algorithms. An autonomous agent can use the compiled knowledge and constrain the number of alternative actions considered by excluding the ones that are likely to be suboptimal.

3 Control of Reasoning: Act or Think

To show the correlation among the amount of information, decision quality, and computation time within our framework, we compare expected utilities of actions arrived at using different amounts of computational resources, and executed with various time delays. Let N be the set of agents and A_i, $i \in N$, be the set of actions of agent i. The expected utility of the best action, α, arrived at using the body of information E and executed at time t, is

$$EU(\alpha^E, t) = \max_{a_i^j \in A_i} \Sigma_k P(S_k|E, t, a_i^j) U(S_k) \qquad (1)$$

where

- $P(S_k|E, t, a_i^j)$ is the probability over the possible outcome states, S_k, given the available evidence E and the action a_i^j executed at time t.
- $U(S_k)$ is the utility of the resulting state S_k.

Having arrived at the best action, the agent i can execute it immediately, or keep on computing by increasing the body of knowledge considered, and including additional information, e, hoping to arrive at better decision. If it were to keep computing with information $E + e$, the result, α^{E+e} would be available for execution at time $t + t'$, where t' is the computation time used. The decision of whether to keep computing with extra information or whether to execute currently best action comes down to the following comparison:

$$EU(\alpha^{E+e}, t + t') > EU(\alpha^E, t) \qquad (2)$$

If the expected utility of possibly higher quality, but delayed, action is greater than the utility of acting on the current best, the agent should keep on computing based on the increased body of information. The challenge is to determine whether the above inequality holds before the computation of α^{E+e} is actually executed. Our approach to flexible decision-making is to use the performance profiles, generated offline, and the notion of value of time, or urgency, to determine whether to act or think. Using performance profiles one can estimate $EU(\alpha^{E+e}, t)$, i.e., the expected value of decision based on information $E + e$, executed currently. Since the actual α^{E+e} is not available, we postulate that the currently available α^E be used to estimate the $EU(\alpha^{E+e}, t + t')$ as follows:

$$EU(\alpha^{E+e}, t + t') = EU(\alpha^{E+e}, t) - [EU(\alpha^E, t) - EU(\alpha^E, t + t')] \qquad (3)$$

In the above, the value of delayed action α^{E+e}, is estimated based on the assumption that its quality deteriorates[1] at the rate similar to that of α^E, which has been computed. Substituting the above to Equation 2 and rearranging terms, we get:

$$EU(\alpha^{E+e}, t) - EU(\alpha^E, t) > EU(\alpha^E, t) - EU(\alpha^E, t + t') \qquad (4)$$

The left hand side of the above inequality is the benefit of further computation, if it can be carried out instantaneously. This quantity can be estimated from the performance profiles. The right hand side is the current estimate of the value of the extra computation time t'. It is closely related to what we define as *urgency*, or the *value of time*, expressed as the rate with which the expected utility of the currently best alternative deteriorates with time. For instance, if the system slowly processes the data and the urgency is high, then the cost of time needed for computation is greater than the benefit of computation, and it is better not to compute further and act immediately. But if the computation time is short, the value of a unit of time is moderate, and the performance profile rises sharply, the extra computation is preferred to immediate action.

4 Performance Profiles of Iterative Deepening Algorithm

Our deliberative decision-theoretic method is the Recursive Modeling Method (RMM). We have implemented a full-blown version of RMM which allows an agent to compute its best action given all that is known about the other agents, about their states of knowledge, and so on. An agent using RMM represents the hierarchy of its modeling knowledge as a nested modeling structure with: (1) the agent's own decision-making situation (represented as a payoff matrix or as an influence diagram), (2) the next level containing the agent's view of other agents' situations (also payoff matrices or equivalent influence diagrams), (3) the deeper level representing the agent's view of how other agents model others, and so forth (as depicted in Figure 1).

The RMM algorithm, operating on the structure represented in Figure 1, can iteratively deepen its scope by including lower modeling levels. Doing so enhances the quality of decision-making as models nested deeper decrease uncertainties about the agents' beliefs about others in a multiagent domain. In the worst case, when none of the simplification methods is applicable, the complexity and run time of decision-making grow exponentially with the number of agents and the level of reasoning. That is why we consider the tradeoff between decision quality and computation time.

Using iterative deepening RMM algorithm, we generate performance profiles, which plot expected output quality versus computation time in proportion to the

[1] In an urgent situation it is better to execute action sooner rather than later. I.e., the expected utility of action decreases as the time passes.

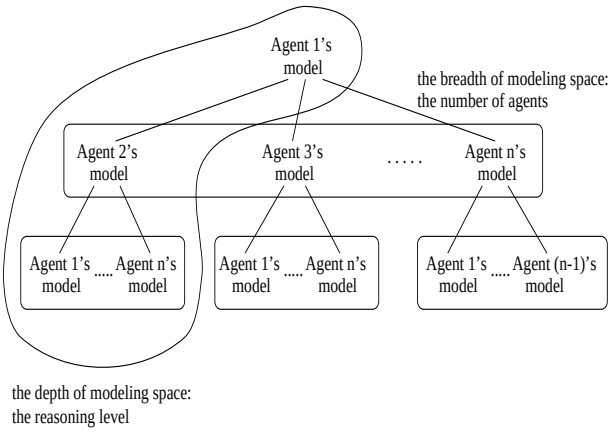


Figure 1: The space of recursively nested models.

breadth and the depth of modeling space. The data for performance profiles is collected offline through simulation of many specific decision-making situations.

Our model of multiagent decision-making under time pressure is the anti-air domain, which consists of a number of attacking targets and coordinating defense units, as depicted in Figure 2. The mission of the defense units is to attempt to intercept the attacking targets so as to minimize damages to own territory. The anti-air defense simulator [Noh and Gmytrasiewicz, 1999a] is written in Common LISP and built on top of the MICE simulator, running on a LINUX platform.

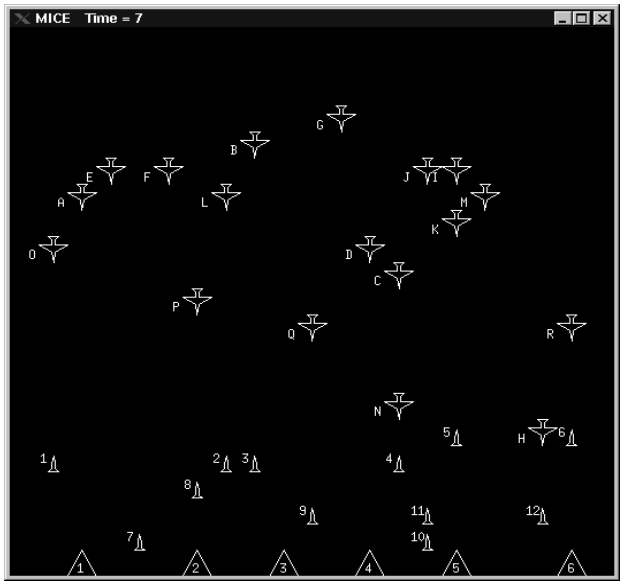


Figure 2: A complex anti-air defense scenario. There are six defense units and 18 incoming targets.

To construct performance profiles using the iterative deepening RMM, we consider six defense agents and six incoming targets.[2] The positions and warhead sizes of targets are randomly generated, and the warhead sizes vary between 100 and 300. We measure the agent's performance in terms of total threat removed. The *total threat removed* is defined as the expected value of a sum of removed warhead sizes of attacking targets. Thus, if a target was chosen for interception, then it contributed ($interception\_probability \times warhead\_size$) to the total threat removed. If a target was not intercepted, it did not contribute at all. The interception probability, $P(H_{ij})$, is dependent on the angle a_{ij} between the target j's direction of motion and the agent i's line of sight, as follows:

$$P(H_{ij}) = e^{-\mu a_{ij}} \quad (5)$$

where μ is an interceptor-specific constant (assumed here to be 0.01).

In the experiment, the breadth of modeling space ranges from two to six and the depth of modeling space varies between one and four. The algorithm begins with the simplest modeling space, that is, two agents and reasoning depth one and incrementally adds the amount of information up to six agents and depth four. For baseline performance, we include the performance of agents controlled by random and independent target selection (i.e., without modeling other agents) strategies. The experimental results of average performance (*total threat removed*) and computation time (*seconds* on a 500 MHz Pentium™ III machine) after 100 runs are summarized into Table 1 as discrete entries. In Table 1, the more total threat removed, the better performance.

The random agents, ignoring all information, showed the worst performance, 516.2, and spent the shortest computation time, 0.013. The performance (527.9) and the computation time (0.017) of the independent agents were similar to those of the random agents. The independent agent, maximizing its own expected utility without considering others, did not score well in this coordination experiment and obtained next to the worst performance. Compared the performance of the independent agent, the performance of the agent coordinating by considering the other neighboring agent, the amount of threat removed, 809.7, was drastically increased up by 53.4%. In this case, we clearly see that modeling other agents is advantageous for effective coordination.

As the depth or the breadth of modeling space increased, the performance of agents was improved and the computation time increased. When the number of agents considered for coordination was two or three, the performance gradually improved; the threat removed from our defense area increased from 809.7 to 859.9 when each defense agent considered the other agents, and increased from 906.2 to 1055.0 when it modeled two other agents. However, as more agents and further reasoning depth were included, the result of performance increase leveled off. The amount of improvement from 3 with depth 4 to

[2] We simplified the scaled-up scenarios for clarity of presentation here. This simplification is made possible by compiled rules we reported on earlier [Noh and Gmytrasiewicz, 1999b].

Table 1: Performance and computation time of the agents using random, independent and iterative deepening algorithm

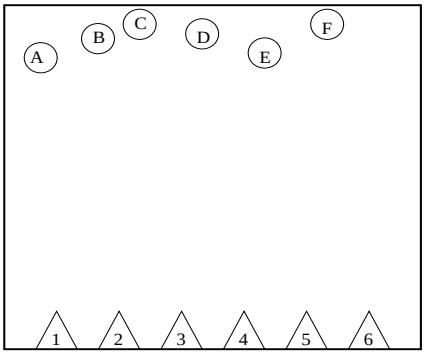
Methods	Performance	Computation Time
Random	516.2 ± 126.6	0.013 ± 0.007
Independent	527.9 ± 175.3	0.017 ± 0.005
2 w/ depth 1	809.7 ± 105.1	0.040 ± 0.015
2 w/ depth 2	816.7 ± 108.9	0.056 ± 0.020
2 w/ depth 3	841.4 ± 82.1	0.058 ± 0.021
2 w/ depth 4	859.9 ± 57.9	0.062 ± 0.043
3 w/ depth 1	906.2 ± 98.3	0.364 ± 0.047
3 w/ depth 2	1001.3 ± 60.2	0.374 ± 0.053
3 w/ depth 3	1050.8 ± 46.3	0.381 ± 0.051
3 w/ depth 4	1055.0 ± 47.9	0.373 ± 0.054
4 w/ depth 1	1055.2 ± 48.9	3.036 ± 0.380
4 w/ depth 2	1055.4 ± 48.0	3.129 ± 0.501
4 w/ depth 3	1055.7 ± 50.8	3.168 ± 0.556
4 w/ depth 4	1055.9 ± 47.4	3.202 ± 0.570
5 w/ depth 1	1057.2 ± 48.0	32.533 ± 11.064
5 w/ depth 2	1057.4 ± 46.6	38.400 ± 7.026
5 w/ depth 3	1057.7 ± 46.3	37.675 ± 6.813
5 w/ depth 4	1058.1 ± 49.0	37.975 ± 7.108
6 w/ depth 1	1060.6 ± 45.3	144.776 ± 97.834
6 w/ depth 2	1060.8 ± 44.4	691.043 ± 166.245
6 w/ depth 3	1061.2 ± 44.3	689.943 ± 164.606
6 w/ depth 4	1061.5 ± 44.0	692.443 ± 164.804

6 with depth 4 in Table 1 was only $6.5 (= 1061.5 - 1055.0)$ due to more information.

As more breadth and more depth were considered, our agents also spent more computation time. However, because of garbage collection in Lisp, the computation time in some cases did not increase in proportion to the reasoning depth: 3 with depth 4, 5 with depth 3, and 6 with depth 3. As more agents were added the run-times ranged over $[0.040-0.062]$, $[0.364-0.381]$, $[3.036-3.202]$, and $[32.533-38.400]$, respectively. When the six agents were considered for coordination, it should be noted that the computation time at reasoning depth two (691.043) took five times as long as that of reasoning depth one (144.776). Proceeding to more than depth one, the agents (6 with depth $2-4$) needed to compute additionally five six-dimensional payoff matrices and so used relatively more runtime to construct and evaluate the payoff matrices. In that the dimension of payoff matrix increased according to the number of agents, the computation time spent depending on the number of agents was saliently different.

5 Determining the Scope of Modeling Using Urgency

Our definition of the urgency, as defined in Equation 4, gets at the intuition that an agent stands to lose utility as time progresses in an urgent situation if the agent is not doing anything to remedy the situation. For example, an air defense unit's not trying to intercept the incoming threats may cause the increase in the probability of getting hit as the time progresses, because the unit's inaction may diminish its chances to successfully intercept the threats. Thus, if the agent's expected utility is inversely proportional to the damages expected in the attack, the situation is urgent in that the defense agent loses utility as time progresses.



Targets: A B C D E F
Positions: (2,3) (5,2) (7,1) (10,2) (13,3) (16,1)

Batteries: 1 2 3 4 5 6
Positions: (3,20) (6,20) (9,20) (12,20) (15,20) (18,20)

Figure 3: A simplified anti-air defense scenario. There are six defense units and six incoming targets.

As a simple example consider the scenario depicted in Figure 3. Assume that the targets proceed two grids down for one second. In this example, we focus on the agent 1's decision-making situation. At the current time the average interception probability is

$$At\ time\ t,\ \Sigma_j P(H_{1j})/6 = 0.9499$$

After one second, the targets are coming closer to the defense agents by two grids, the angles between the battery's line of sight and the targets increase, and the interception probability from the agent 1's perspective decreases as follows:

$$At\ \ time\ \ t+1,\ \Sigma_j P(H_{1j})/6 = 0.9483$$

In our settings as described in the previous section, the average total warhead size is $1,200 (= 200 \times 6)$. Given the interception probabilities at different time steps, the average total warhead size and the computation time, the urgency, or the cost of computation, is

$$\begin{aligned} EU(\alpha^E, t) - EU(\alpha^E, t + 0.017) &= \\ (0.9499 - 0.9483) \times 1,200 \times 0.017 &= 0.0326 \end{aligned}$$

Given the performance profiles in Table 1, the benefit of computation (the left hand side of Equation 4) is

$$EU(\alpha^{E+e}, t) - EU(\alpha^E, t) = 527.9 - 516.2 = 11.7$$

In this example scenario, therefore, since the benefit of computation is greater than the cost of computation, the agent should not take the immediately available action α of shooting at a randomly picked target at the initial time t, but it should consider the extra information

e (consisting of the agent's own decision-making situation), compute $EU(\alpha^{E+e}, t+t')$, and be ready to execute the best action at time $t + 0.017$.

It turns out that the break-even point, at which the agent should cease computation, is to model 3 agents with depth 3 in the example scenario. If the parameters of the particular situation at hand change, however, say when the missiles move faster, they are positioned closer, or if the computation is slower, the optimal scope of reasoning is lower and less computation will take place. In general, the factors of the calculation of value of time include the capability of data processing machine and the external factors imparting urgency to the situation.

We tested the notion of urgency in the anti-air defense settings which contain six defense units and six incoming targets. To identify the appropriate scope of modeling given the offline pre-computed performance profiles (Table 1), each defense agent calculated the benefit and the cost of computation for a level. As a result, after 100 runs, the defense agents stopped reasoning when the average breadth of modeling space was 3 and the average depth of modeling space was 3.017. That is, they considered the adjacent two agents with the reasoning depth just over three, on the average, for their coordinated decision-making. When the agents were under time pressure, thus, they could decide the proper modeling space by calculating their urgency, and reduce the scope and processing time to react faster.

6 Conclusions

The iterative deepening algorithm can be a useful tool that generates partial results according to the breadth and the depth of the modeling space. We generated offline performance profiles by using the iterative deepening algorithm in an anti-air defense domain. We think that the performance profiles for other coordination domains may be similar, but should be obtained separately offline. As the amount of breadth and depth of modeling information increased, the quality of decision was improved. We also extended our multiagent interaction model by introducing the concept of urgency. This notion presents the value of time in a specific situation, when decisions have to be made under time pressure and uncertainty. We integrated this concept into the flexible decision-making framework for identifying the proper scope of modeling information. The most promising aspect of this method is that the reasonable model structure can be employed to avoid wasteful computations and produce a practical decision during real-time.

We experimented with the iterative deepening algorithm and the concept of urgency in the anti-air defense domain to assess agent's decision-making quality. Anti-air defense is certainly a real-time task, and any overhead or loss of timely response can result in the destruction of coordinating agents and areas they are defending. The iterative deepening algorithm and the calculation of value of time prevented catastrophic failure and provided the best decisions in the high-stakes anti-air defense.

We will address the problem of what and how much offline performance profiles are necessary to gain the real-time advantage. For a more systematic evaluation of the framework, the performance profiles using the iterative deepening algorithm will be generated in various domains. In parallel, when insufficient data is available for offline simulation, we will test whether or not our framework produces deterioration in performance of the real-time deliberation.

Another problem to be answered is that we need to validate how each flexible decision-making method proposed can be integrated into a whole procedure. Even though we already tested whether the respective methods can be successfully applied to the time critical situation, we still need to experiment with the whole procedure selectively combined for various situations.

References

[Dean, 1990] T. Dean. Decision-theoretic control of inference for time-critical applications. *Artificial Intelligence*, pages 1–28, November 1990.

[Durfee, 1999] E. H. Durfee. Practically coordinating. *AI Magazine*, 20(1):99–116, 1999.

[Gmytrasiewicz and Durfee, 2000] P. J. Gmytrasiewicz and E. H. Durfee. Rational coordination in multi-agent environments. *Autonomous Agents and Multiagent Systems Journal*, 3:319–350, 2000.

[Gmytrasiewicz *et al.*, 1998] P. J. Gmytrasiewicz, S. Noh, and T. Kellogg. Bayesian update of recusive agent models. *User Modeling and User-Adapted Interaction: An International Journal*, 8(1/2):49–69, 1998.

[Good, 1971] I. J. Good. Twenty-seven principles of rationality. In V. P. Godambe and D. A. Sprott, editors, *Foundations of Statistical Inference*, pages 108–141. Holt, Rinehart, and Winston, 1971.

[Horvitz, 1988] E. J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings of the 7th AAAI*, pages 111–116, August 1988.

[Lesser, 1998] V. R. Lesser. Reflections on the nature of multi-agent coordination and its implications for an agent architecture. *Autonomous Agents and Multi-Agent Systems*, 1(1):89–111, 1998.

[Noh and Gmytrasiewicz, 1999a] S. Noh and P. J. Gmytrasiewicz. Implementation and evaluation of rational communicative behavior in coordinated defense. In *Proceedings of the 3rd Autonomous Agents*, pages 123–130, May 1999.

[Noh and Gmytrasiewicz, 1999b] S. Noh and P. J. Gmytrasiewicz. Towards flexible multi-agent decision-making under time pressure. In *Proceedings of the 16th IJCAI*, pages 492–498, August 1999.

[Russell and Wefald, 1991] S. J. Russell and E. H. Wefald. Principles of metareasoning. *Artificial Intelligence*, 49:361–395, 1991.

[Vidal and Durfee, 1995] J. Vidal and E. Durfee. Recursive agent modeling using limited rationality. In *Proceedings of the 1st ICMAS*, June 1995.

[Zilberstein and Russell, 1996] S. Zilberstein and S. J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1):181–213, 1996.

MULTI-AGENT SYSTEMS

LIFELIKE CHARACTERS

A Layered Brain Architecture for Synthetic Creatures

Damian Isla naimad@media.mit.edu
Robert Burke rob@media.mit.edu
Marc Downie marcd@media.mit.edu
Bruce Blumberg bruce@media.mit.edu

The Media Laboratory
Massachusetts Institute of Technology
20 Ames St.
Cambridge, MA 02139

Abstract

This paper describes a new layered brain architecture for simulated autonomous and semi-autonomous creatures that inhabit graphical worlds. The main feature of the brain is its division into distinct systems, which communicate through common access to an internal mental blackboard. The brain was designed to encourage experimentation with various systems and architectures. It has so far proven flexible enough to accommodate research advancing in a number of different directions by a small team of researchers.

1.0 Introduction

One approach to designing intelligent systems is to look to existing biological systems for clues and design principles. This paper describes C4, the latest in a series of brain architectures built on this principle by our group. This architecture is used for simulating the behavior of agents, or *creatures*, that inhabit a graphical world. These creatures are able to sense their environment, learn appropriate actions based on expectation of external reward and navigate their environment. C4 sets up a framework to support these and many other mental abilities. The content and structure of this framework are inspired by the abilities of real animals and attempt to deal with some of the constraints that they face.

Some of the goals for the system were completely practical: it needed to support graphics with at least a 20Hz frame rate, input devices (mice and microphones as well as more exotic interfaces) and network rendering. It also needed to be scalable enough to support a reasonable number of autonomous creatures all sensing and reacting to each other and the world.

But more importantly, the architecture needed to facilitate the construction and control of synthetic creatures. Intelligence is often seen as the confluent effect of many individually unintelligent components (as in [Minsky 1985]) and the architecture needed not only to support those components individually but also to allow them to coexist and communicate coherently within one brain. Therefore the primary goal was to build a system that facilitated:

Figure 1: Duncan the Highland Terrier

- Reactive behavior: it should be easy to design and implement the kind of reactive behavior that many previous works in the field of autonomous agents support. [Brooks 1991, Tu et al 1993, Blumberg 1995, Perlin 1996, Yoon 2000];
- Learning: creatures should adapt their behavior based on reward and punishment feedback from the world;
- Extensibility: The architecture should be easily extensible in order to support research in various different directions by different researchers.

The result is a highly modular architecture with few essential subsystems and many opportunities for expansion. The canonical brain includes an internal blackboard, Sensory and Perception Systems, Working Memory (a short-term memory model), and Action, Navigation and Motor Systems.

We have implemented two significant projects to date with C4. One project is *sheep|dog* (**Figure 1**), an interactive installation piece in which a user plays the role of a shepherd who must interact through a series of vocal commands with Duncan, a virtual sheepdog, to herd a flock of sheep. This system demonstrated some of the basic reactive, perceptual and spatial abilities of the creatures built under C4. The other project is *Clicker*, in which the user trains Duncan to perform a variety of tricks using the same "clicker training" technique used to train real dogs.

The paper proceeds as follows: Section 2.0 will outline the world model being used; section 3.0 will delve into the architecture of the brain itself, and the make-up and function of the various systems that comprise C4; section 4.0 will present the results of building C4 and the installations that use it; section 5.0 will present a discussion of various successful design elements of C4 and sections 6.0 and 7.0 will describe some future and related work.

2.0 World Model

A formal abstraction exists between the agent and the world. The World model's primary function is to maintain the list of creatures and objects and to act as an event blackboard for the posting and distribution of world events. It also coordinates network synchronization and manages rendering.

World events take the form of *DataRecords*, perceptual nuggets that can be processed by a creature's Sensory and Perception Systems. A DataRecord can represent anything from an acoustic pattern (Sheep|Dog allowed the user to "speak" to the dog through a microphone interface) to a visual or symbolic event. Whether a specific form of DataRecord can be interpreted depends entirely on the sensory and perceptual abilities of the sensing creature.

In a single execution of the update loop, events are fed to each of the creatures and objects and each is prompted to update itself (this process can happen in parallel). Most creatures and objects will themselves produce events that the World will harvest and make available in the next timestep. Pre- and post-update tasks, including UI updating, network coordination and rendering, are also performed.

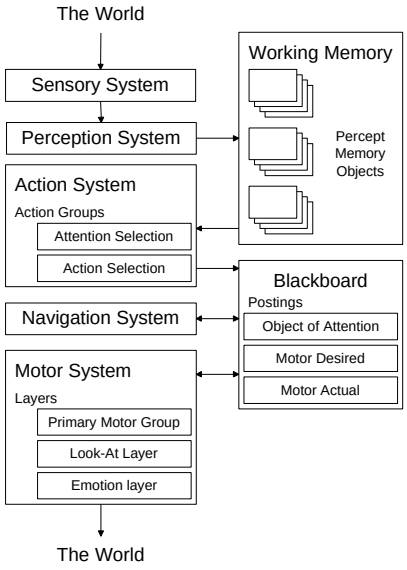


Figure 2: The brain architecture

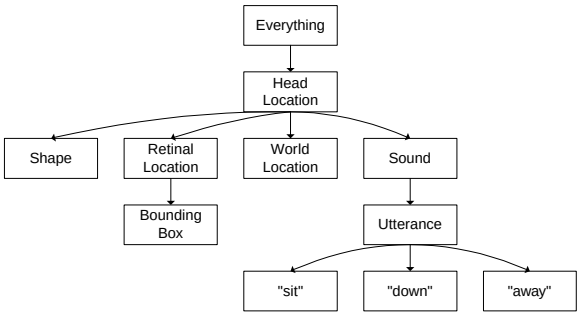


Figure 3: The Percept Tree. The Head Location Percept is derived from the root of the tree because, at present, all events have locality. Retinal location and shape only apply to visual events, just as an event can only be classified as an utterance or a specific utterance if it is an auditory event.

3.0 Brain Overview

Figure 2 shows the layout of a typical creature's brain. C4 is organized into a collection of discrete systems that communicate through an internal blackboard. Any part of the brain can write to or read from specific "slots" in this blackboard (differentiated by string labels).

A detailed description of each system follows.

3.1 Sensory System

Physical intelligent systems (biological or otherwise) by necessity feature a set of well-defined sensors through which information about world-state and world-events pass. We protect the integrity of this world-agent division by making use of a Sensory System abstraction.

The Sensory System is a filter through which all world-events (represented by DataRecords) must pass. Unlike its physical equivalents, where any data that passes through is fair game, in C4 the Sensory System plays an active role in keeping the virtual creature's virtual sensation honest. In a simulated world, there is potentially much more accessible information than the creature, limited by its sensory apparatus, should be able to sense. For example, though Duncan the sheepdog *could* be given the location of the sheep that is behind him, he *shouldn't* be given it. While often the role of the Sensory System is to filter out data that can not be sensed, other times its role is to transform it. For example, it converts visual location information into the local space of the sensing creature (Duncan receives all location information from the world in the coordinate frame of his left eye) or in attenuating the intensity of a sound or acoustic event proportionally to the square of the distance from the source. It is because the same piece of data will be viewed differently by each creature in the world (and because sometimes we want some creatures to be able to cheat and not others) that the individual creature's Sensory System performs the event filtering rather than the world itself.

Perceptual honesty is an important theme for our research, since we feel that more honest perception leads to more believable behavior. More importantly, creatures in the real world are able to make generally good decisions despite noisy, unreliable and occasionally entirely *missing*

data. This is the point of Brooks' insistence on *situatedness* [Brooks 1991]. While we do not claim that our virtual world holds anything close to the complexity of a physical environment, the Sensory System allows us to begin to face some of the problems like noisy sensation and McCallum's *perceptual aliasing* [McCallum 1995].

3.2 Perception System

Once the stimulus from the world has been "sensed" it can then be "perceived." The distinction between sensing and perceiving is important. A creature may "sense" an acoustic event, but it is up to the Perception System to recognize and process the event as something that has meaning to the creature. Thus, it is within the Perception System that "meaning" is assigned to events in the world.

The Perception System takes the form of a *Percept Tree* (**Figure 3**). A *Percept* is an atomic classification and data extraction unit that models some aspect of the sensory inputs passed in by the Sensory System. Given a DataRecord it returns both a match probability (the SheepShapePercept will return the probability that a DataRecord represents the experience of seeing a sheep) and, if the match is above a threshold, a piece of extracted data (such as body-space coordinates of the sheep). The details of how the confidence is computed and what exact data is extracted are left to the individual percept. The percept structure might encapsulate a neural net or it might encapsulate a simple "if … then …else" clause. This freedom of form is one of the keys to making the C4 Perception System extensible, since the system makes no assumptions about what a percept will detect, what type of data it will extract or how it will be implemented.

Percepts are organized hierarchically in terms of their specificity. For example, a ShapePercept will activate on the presence of any kind of shape whereas one of its children may activate only on a specific type of shape (e.g. a SheepShapePercept). The children of a percept will receive only the data that was extracted by its parent to process. This hierarchical structure is primarily an efficiency mechanism (no point in testing whether an event is the spoken word "sit" if it has already been determined that the event was not an acoustic one) and is very similar to previous hierarchy-of-sensors approaches.

Many percepts are plastic, using statistical models to characterize and refine their response properties. These percepts can not only modulate their "receptive fields" (the space of inputs to which they will respond positively), but, in concert with the Action System, can modify the topology of the tree itself, dynamically growing a hierarchy of children in a process called *innovation*. As will be described in section 3.5, the process of innovation is reward-driven, with only percepts that are believed to be correlated with increasing the reliability of an action in producing a desirable outcome being prompted to innovate.

Both the confidence and the extracted data of every percept are cached in a *PerceptMemory* object. When a DataRecord enters the Perception System, a new PerceptMemory is created, and as the DataRecord is pushed through the Percept Tree, each percept that registers a positive match adds its respective data to the new PerceptMemory. Thus, given a sensory stimulus, the PerceptMemory represents all the creature can *know* about that stimulus.

3.3 Working Memory

Like other agent-control architectures (e.g. [Rosenbloom et al. 1993]), C4 contains a *Working Memory* structure whose function mirrors that of the pyschological conception of Working Memory – an object-based memory that contains information about the immediate task or context. The ultimate goal of Working Memory is to provide a sensory history of objects in the world. It is on the basis of these histories that action-decisions will be made, internal credit for reward assigned, motor-movements modulated, etc. In C4, Working Memory is a repository for persistent PerceptMemory objects. Taken together, they constitute the creature's "view" of the world.

The PerceptMemory is itself a useful structure. By caching together the various perceptual impressions made by a world event ("the thing that was in front of me was also blue," blueness and relative location being separate Percepts) they solve (or perhaps avoid) the infamous *perceptual binding problem* ([Treisman 1998]). They also allow us to submit complex queries to WorkingMemory: "which is the sheep that is nearest me?"

PerceptMemory objects become even more useful when they incorporate a time dimension with the data they contain. On any one timestep, the PerceptMemory objects that come out of the Perception System will by necessity only contain information gathered in that timestep. However, as events often extend through time, it is possible to *match* PerceptMemory objects from previous timesteps. Thus a recent visual event may represent only the latest sighting of an object that we have been tracking for some time. In the case of visual events, matching is done on the basis of shape, or on the basis of location when shape is not enough to disambiguate incoming visual events (e.g. distinguishing between two sheep). Location matching is also used to combine perception information of different modalities, for example a visual event and an auditory one that originate from the same location. Assuming that a match is found, the data from the new PerceptMemory is added to the history being kept in the old one.

The new *confidence* is also added to a history of confidences. On timesteps in which percept data is not observed, the confidence in the existing data is decayed. The rate of decay is in part determined by the percept. Roughly, the rate of decay should be proportional to the observed variability of the data.

3.4 Prediction and Surprise

The "view" that Working Memory provides of the world can be informed by more than just direct perception. Much like our own perception, where unobserved data is subconsciously "filled in" by low level predictions and assumptions, creatures implemented under C4 can be designed to act upon events that are likely to occur, or qualities that objects are likely to have. We believe that the ability to recognize temporal patterns and act on them is an essential component of common sense intelligence.

Sometimes prediction is used not to predict the future but simply to maintain a coherent view of the present. The stream of sensory data coming from an object can easily be interrupted – for example, because the object is out of the creature's visual field, or because it is occluded by another object. In these cases, prediction can allow a creature to maintain a reasonable estimate of where the object is, even though it is not being observed.

The actual mechanisms of prediction can take many forms. Since PerceptMemory objects contain histories of percept data, it is possible, if the data are vector or scalar, to use function approximation techniques to extrapolate values. In more complex cases, periodic behaviors or percept-activity correlations (the bell always rings before the food appears – classical conditioning, essentially) can be recognized and exploited. These prediction-mechanisms could conceivably extend to common sense knowledge about the world – if an object is suspended with no visible support, it might be predicted to fall.

The occasional deviation of predictions from the actual state of the world – and the magnitude of that deviation – also provide a basis for *surprise*. Surprise is an excellent method for focusing perception, and a PerceptMemory which has just provided a surprising stimulus is an excellent candidate for the creature's object of attention. [Kline 1999] includes an excellent discussion of expectation and surprise in synthetic characters. C4 does not currently make use of surprise.

3.5 Action System

Given a set of PerceptMemory objects that detail the perceived state of the world, the creature must decide what action(s) it is appropriate to perform. Many types of decision-making processes are possible here, however, this section will discuss the action system that we have implemented in C4.

3.5.1 ActionTuples

Any Action-representation must address a number of fundamental questions, namely:

- What do I do?
- When do I do it?
- What do I do it *to*?
- How long do I do it for?
- What is it worth?

Our representation of action, the *ActionTuple*, directly addresses these five questions through its five subcomponents:

Primitive Action(s) (what do I do?): A piece of code that actually executes the action in question. An ActionTuple's set of primitive actions typically modify the contents of the blackboard. These postings in turn act as messages to other parts of the system. For example, the MOTOR_DESIRED posting holds the name of a physical behavior requested of the Motor or Navigation Systems.

TriggerContext (when do I do it?): A piece of code that returns a scalar value representing the relevance of an ActionTuple given the current state of Working Memory. Triggers are typically references to percepts in the Percept Tree (a trigger that points to the "Sheep Shape" percept will return a high relevance given any PerceptMemory that has a high "Sheep Shape" confidence). However, the TriggerContext is general enough that more complex trigger-conditions can be hand-crafted. As we will see, Percept-based triggers are useful because they can be automatically generated through the learning process.

ObjectContext (what do I do it to?): A piece of code that chooses a target for the action. Again, it is often defined in terms of percepts ("perform the action on something that is sheep-shaped AND blue"). When an ActionTuple is active, the ObjectContext posts the PerceptMemory chosen into the OBJECT_OF_ATTENTION posting of the internal blackboard, thereby making it available to the rest of the system. The ObjectContext is an optional component, since not all actions are necessarily targeted.

DoUntilContext (how long do I do it for?): A piece of code that returns a scalar representing the continuing relevance of an ActionTuple while it is active. This could take the form of a timer, which drops to zero after a specific period of time, or some code that looks for more complicated ending conditions in Working Memory.

Intrinsic Value (what is it worth?): ActionTuples are ascribed an *intrinsic value*, which is an indicator of how generally "good" the ActionTuple is. This is similar to the Q-value in Q-learning (see [Ballard 1997]). This value can be used to bias the Action-Selection process and can be modified through learning (see below).

3.5.2 Action-Selection

The intrinsic value and the relevance (as determined by the Trigger or the DoUntil contexts) can be combined into a single *evaluated value* corresponding to the amount of reward expected to result from performing an action. ActionTuples can then compete for expression on the basis of this evaluated value.

ActionTuples are grouped into *ActionGroups* that are responsible for deciding at each moment which single ActionTuple will execute. Each ActionGroup can have a unique Action-Selection scheme, but the most common scheme is described below.

When ActionTuples are added to an ActionGroup they may be placed on either the *Startle* or the *Default* tuple list. The *Startle* list contains high priority ActionTuples that compete deterministically, i.e. the one with the highest non-zero evaluated-value wins. If no ActionTuple on the Startle list is relevant, the ActionTuples on the Default list are allowed to compete probabilistically for expression on the basis of their *e-values*.

If an ActionTuple is active, it is generally allowed to stay active until its DoUntil condition is met. When it is met, the selection process takes place again. There are two cases in which an active ActionTuple can be interrupted: if a StartleTuple becomes relevant (or more relevant than the current one, if the current one is also a startle) or if the world changes significantly. This change is measured in terms of the evaluated values of inactive tuples: if an inactive ActionTuple's evaluated value has more than doubled recently, then another probabilistic tuple selection takes place between that tuple and the current one.

In C4's canonical Action System, there are two main ActionGroups. These are (in order of execution):

- *AttentionGroup*: Chooses the creature's focus of attention (e.g. things that are large, things that are moving fast, etc.). This decision will often be overridden by later-executed actions.
- *Primary ActionGroup*: The ActionGroup whose actions determine large-scale body motion.

3.5.3 Learning in the Action System

Learning is an important focus of our research, particularly the kind of learning that is observed in animals. The Action System implements three types of learning that together allow creatures to be trained in a manner similar to that used to train real dogs ([Wilkes, personal communication]).

Credit Assignment: When a new ActionTuple with a high intrinsic value is activated (for example, the startle tuple "eat", which can only be run in the presence of food) we wish to give credit to the action that led to that tuple being activated. This back-propagation of value scheme is very similar to that seen in temporal difference learning [Sutton 1998]. However, unlike classical temporal difference learning, it is not always the tuple that actually ran that is given credit – it can be a tuple with a similar primitive action list with a more specific trigger that was relevant at the time. This choice is based on a combination of reliability and novelty metrics (the length of the paper precludes going into too much detail). Whichever tuple is ultimately given credit, some percentage of the intrinsic value of the newly-activated tuple is added to the intrinsic value of the credited tuple.

State-space discovery: When Duncan the sheepdog is rewarded for sitting in response to the utterance "sit", there are two conclusions he can make: first, that sitting in response to "sit" is a good idea; second, the specific acoustic pattern that was reacted to must be a good example of the utterance "sit." Since the "sit" classifier (in the form of a percept in the Perception System) is actually represented by an example-based statistical model, we can use reward information to update that model. When Duncan receives a large amount of reward for responding to an utterance, that utterance is added back into the statistical model that recognized it in the first place. Thus at the same time as Duncan learns the value of actions, he refines his conception of the appropriate *context* for those actions.

Innovation: Most ActionTuple triggers are direct references to percepts in the percept tree. When they are, the triggers keep statistics about the reliability of the percept's *children* in predicting reward. The hope is that one of the children of the current trigger is actually a better predictor. For example, sitting in response to "Any Utterance" (assume "Any Utterance" is a percept with several children) may be quite a good thing, but sometimes fails entirely to procure reward. By examining the reliability statistics, however, we may discover that sitting in response to a "sit utterance" (assume a "sit utterance" percept is a child of "Any utterance") is far better and more reliable. If this is the case, the ActionTuple in question will create a copy of itself, replacing its trigger with a new trigger that references the "sit utterance" percept. This new ActionTuple will be added as a "child" to the old one, and will supercede its parent when both are relevant in an action selection round. This approach was inspired, in part, by [Drescher 1991] who proposed a very similar scheme for exploring what are essentially state-action pairs.

Innovation can also be prompted in the percept tree. Some forms of statistical models allow hierarchical classifications to be grown (for example, hierarchical clustering algorithms). If an ActionTuple that references such a classifier decides to innovate, it prompts the classifier also to innovate. How this mechanism works ultimately depends on the kind of model the percept contains (clustering algorithms can isolate subclusters and spawn new percepts to represent them). Thus reward feedback can also drive the growth of the percept tree.

Both state-space discovery and innovation illustrate the intimate coupling between perception and action selection. *The creature only bothers to refine senses that ultimately allow it to make better decisions about what to do*.

3.6 Navigation System

The deceptively simple act of "eating the food" involves a host of problems: the creature must be near the food, be oriented toward it and if approaching it is necessary, must avoid physical obstacles on the way. The Navigation System allows such spatial competencies to be included implicitly in the Action System's high-level behaviors.

The Navigation System typically functions by overriding the motor commands passed down by the Action System. In some cases this command is for an explicit Navigation task, such as "APPROACH." In other cases, the

command is directed to the Motor System but with extra approach and orientation conditions specified which the Navigation System must work to satisfy. In either case, the original decision of the Action System is overridden with a more immediately appropriate motor command. If the Action System requests an "APPROACH", the Navigation System might decide that the best way to implement that request is through a "GALLOP". If the Action System requests a "BEG" but the Navigation System is instructed to orient the creature first, that command might be replaced with a "TURN". The "BEG" will continue to be overridden until the orientation condition is satisfied.

The Navigation System allows a convenient level of representation in the Action System, because it relieves the Action System of the burden of implementing the decisions it makes. "Approaching" may indeed precede each "eating", but behaviorally, both should be part of a single "eating" act – especially from the point of view of any learning that takes place. Ultimately, the majority of animal behaviors follow the "approach, orient and do" model, and the Navigation System allows these behaviors to be represented with high-level atoms.

3.7 Motor System

Ultimately, the primary output of a virtual creature is motion, whether it be motion for the purposes of locomotion, gesticulation or expressivity. This lowest level – the level of joint-angle control of the transform hierarchy that represents the body of the creature – is controlled by the Motor System. This system takes its inputs from MOTOR_DESIRED and MOTOR_ADVERB entries of the internal blackboard.

The design of the Motor System was inspired by the work of Rose et al [Rose 1999]. The system is organized as a *Verb Graph* in which the nodes represent hand-made animations (Verbs) and the edges represent allowed transitions between Verbs. The Verb Graph in this way represents some of the basic physical and continuity constraints of having a body. Multiple labeled examples of the same Verb may be provided that span an *Adverb space*. If multiple examples are provided, the Motor System does multi-target interpolation at runtime based on blend coefficients provided externally by the MOTOR_ADVERB entry of the blackboard. For example, examples of left-, straight- and right-walks are used to create a continuous space of directional walks.

Layering as discussed in [Perlin et al. 1996] is also supported by the Motor System. Layering allows multiple non-conflicting animations to be played concurrently (for example, walking and waving hand). Duncan the sheepdog has a number of layers corresponding to body-pose, head-layer (for look-ats), tail-layer etc.

Space limitations preclude a full discussion of recent research in motor control, which has centered around an extension of the Verb Graph system called a *Pose-Graph*. Pose-Graphs allow source animation files to be decomposed into atomic animation-chunks which can then be connected into a directed graph through a "pose-space". Creatures can then use this graph to explore new animations, potentially adding these new animations to its existing list of motor skills. This system can also be used to demonstrate simple and complex *shaping*, a training technique through which motor skills are perfected through successive approximations, and *luring*, in which a trainer can lure a creature into a certain pose and then reward that pose. For a complete discussion of the Pose-Graph motor system and its integration into learning, see [Downie 2001].

4.0 Results

Sheep|dog was created to demonstrate some of the basic abilities of the creatures implemented under C4. The project showed creatures acting and reacting to each other and the world. It also employed some of the group's acoustic pattern recognition research to allow Duncan to classify user utterances as one of six possible commands. This classification could be trained through a "one-shot learning" interface so that a new user could achieve a high recognition rate after a very short (about 15 seconds) training routine.

The project also served as a stress test for the system's engineering. It featured two creatures with full brains (Duncan and the shepherd) and six sheep (flocking according to Reynolds' *BOIDS* algorithm [Reynolds 1987]) and a number of world-obstacles, all running scaled-down versions of C4. The project also featured distributed rendering, with two clones running subsets of the system rendering the same world from different views.

The learning algorithms being developed by our group were put to use in *Clicker*, in which a user can train Duncan using "clicker training" – an actual dog-training technique in which behaviors are "marked" (by a salient click sound) and then reinforced with food reward. In this simulation, Duncan can be trained to associate vocal commands with behaviors, and demonstrates a number of the phenomena that one sees in real dog training (i.e. Thorndike's Law of Effect, shaping, resistance to extinction etc.). Given an initial repertoire of a dozen basic behaviors (e.g. "sit", "shake", "lie-down", "beg", "jump", "go-out") together with basic navigational and behavioral competencies we have been able to train him to respond to both symbolic gestures (i.e. game-pad button presses), and more significantly to arbitrary acoustic patterns (indeed one user trained Duncan to respond to commands in Gha, the language of Ghana). A dozen such tricks can be trained in real-time within the space of 15 minutes. We have also demonstrated simple shaping with both motor systems and complex shaping and luring with the Pose-Graph based motor system

5.0 Discussion

5.1 Heterogeneous Design

Heterogeneity of decision-policy and representation is seen at multiple levels in C4. The Motor System uses the Verb

Graph structure to plan movements. The Action System makes decisions using ActionTuples. Within the Action System itself, different decision-making policies are employed – Startle ActionTuples are treated differently from Default ActionTuples. This variety, we feel, contributes to the system's robustness and generality. The many kinds of problems that animals can solve entail many kinds of solutions. In order to be a viable platform for intelligence in all its multitude of forms, C4 needed to support and encourage this heterogeneity.

In acting as the go-between for these disparate systems and representations, the creature's internal blackboard plays an important role. This generic communication device allows the easy reordering or omission of entire systems as well as the overriding of the output of one system by another in a way that would be very difficult if input and output pairs were directly coupled.

5.2 Simulation- vs. Mental-Representations

C4 enforces a strict split between simulation representations and mental representations. The former constitute the "ground truth" of the Virtual World, and are used, for example, in generating the graphics output. The latter are the PerceptMemory objects. Forcing a creature to act strictly on the contents of its own memory demands that it make decisions not on the basis of the world's state, but on the basis of its *view* of that state.

By divorcing these two representations, many "second-level" effects become possible, most of them arising from situations in which the two representations fail to match, i.e. mistakes. These effects include mistaken identity, surprise, confusion and the ability to be teased. Paradoxically, these "mistakes" can add greatly to a creature's realism. They also provide some insight into how real creatures commit and recover from these kinds of errors.

5.3 Supersumption

Two architectural themes are seen throughout C4. The first, reminiscent of Brooks' *subsumption*, is when high-level systems send control signals to low-level systems in order to change their behavior (in this case "high-level" and "low-level" are not intended to reflect "degree of sophistication"). Another technique for control is when the decisions of high-level systems, which have a good general idea of *what* to do, are overridden by specialized low-level systems that have specific knowledge of *how* to do it. We call this *supersumption*.

This technique is seen in the Action System, when an object of attention chosen by the Attention Group is overridden by an object of attention chosen by the Primary Action Group (since it controls overall action, it probably has a more appropriate choice). It is seen again in the Navigation System, where motor commands sent by the Action System are overridden with more immediately appropriate motor commands. In this case, the Motor System does not know where its instructions come from and the Action System need not concern itself with the details of how its instructions are implemented.

5.4 Easy Behavior Design

Throughout C4, there is much machinery to make it easy for creature designers to specify behavior. Believable behavior involves many details: the creature's gaze and physical positioning must be controlled as appropriate for the active behavior, the creature's physical emotion-layers must reflect an attitude toward the behavior or the behavior's target, etc. One strength of C4 is the system's ability to handle many of these details automatically through mechanisms like the Attention System (which also directs eye gaze), the Navigation System (which hides behavior implementation details from the Action System and the designer) and layers and adverbs in the Motor System. It remains for the designer to fill in a few critical components of the behavior, such as specifying triggers and ObjectContexts. The extensibility of the system also makes it clear where and how a creature's brain needs to be expanded when new abilities are added. (Perhaps the Perception System needs a percept to detect a new type of world event, or perhaps the Motor System needs a new motor skill.) Whatever the case, the system conspires to make *simple things simple and complex things possible*.

5.5 Building the entire system

No actual brain center ever functions in isolation, but instead feeds and is fed by a myriad of other centers. Intelligent behavior is the combination of all their effects. C4, through its extensibility and easy reconfigurability allows us to explore some of the basic interactions between the various components of the brain. This is why building the *entire* creature is critical.

6.0 Future work

C4 continues to be a work in progress. Work over the next year will continue to emphasize behavioral adaptation and motor learning. We are also exploring how to model development (physical and mental) and social behavior. Indeed, C4 is being used as the behavioral engine for a simulated wolf pack. We are also integrating a model of hippocampal spatial learning using landmark and path-integration navigation that will soon give the creatures a truer sense of space. Inspired by the work of Gallistel [Gallistel 2000], we are continuing to formalize our ideas about time, rate and conditioning.

7.0 Related Work

Our work borrows heavily from the impressive work that has come before. Our ideas about super and subsumption between layers follow from Brooks' work (e.g., [Brooks 1991]) as does our emphasis on building the whole creature. However, in contrast to Brooks' work, our layers communicate through Working Memory which seems to us

to be a more general approach and one which reflects our belief that it is difficult to impose a strict layering in practice. Our emphasis on the value of taking an ethologically inspired approach follows from [Reynolds 1987, Tu 1994, Blumberg 1995, Yoon 2000], whose behavior systems were inspired by the behavioral models of ethologists such as Tindbergen and Lorenz. We go beyond this work in our representation of action that admits "time and rate" as first class objects, and in our integration of learning. In addition, our approach to perception, from the Percept Tree to PerceptMemory Objects, is a good deal more sophisticated and powerful than that proposed in these earlier systems. We also allow a "behavior designer" to work at a higher level of abstraction and one which is more familiar to most people. The tangible benefit of this approach is that it is far easier to develop creatures using our system than in our previous work. For example, the creatures described in Yoon 2000 typically required 30-40 pages of source code to specify their behavior systems. By contrast, Duncan is specified in less than 10 pages of source. Finally, our action selection mechanism balances the pragmatic need for both deterministic and probabilistic choice of action. Our representation of action sits between that typically used in reactive systems and that used in planning systems. As such it borrows from the work of Maes and Firby [Maes 1990, Firby 1992] and in its emphasis on choosing a few good representations, from Minsky [Minsky 1985]. However, by focusing on the 5 big questions (when, to whom, what, for how long, and how much is it worth), the ActionTuple goes beyond these previous representations in providing a surprisingly powerful and intuitive representation for action. Our system also is novel in showing how learning may be integrated into this representation. Our approach to learning borrows ideas from traditional reinforcement learning [Ballard 1997 for a review], animal psychology [Gallistel 2000] and dog training [Gary Wilkes, personal communication]. Our approach to innovation is directly inspired by Drescher's seminal work [Drescher 1991]. While only touched on briefly in this paper, our system is novel in its ability to perform state-space discovery (i.e. learning new percepts), behavioral adaptation (i.e. learning new ActionTuples), and motor learning (i.e. learning new motor actions) in an integrated framework. Through the use of heuristics such as temporal proximity, simple statistics such as reliability and novelty, and by using the consequences of actions to help discriminate between good and bad examples from which to build models of relevant state, our system provides an interesting example of how learning may be successfully and powerfully integrated into a larger behavioral framework. While our system does not do "cognitive modeling" as proposed by Funge [Funge 1999], the system described could easily be integrated into Funge's architecture. Our motor system design borrows heavily from the ideas of Rose [Rose 1999] and Perlin [Perlin 1996]. Our contribution, particularly with respect to Rose, is to demonstrate the usefulness of his approach.

Acknowledgements

Many thanks to everyone in the Synthetic Characters group who worked hard on C4 and Sheep|Dog: Scott Eaton, Yuri Ivanov, Ben Resner, Bill Tomlinson, Matt Berlin, Jesse Gray and Geoff Beatty.

References

[Ballard 1997] Ballard, D., An Introduction to Natural Computation, MIT Press, Cambridge 1997.

[Blumberg et al. 1995] Blumberg, B.M., Gaylean, T., Multi-level Direction of Autonomous Creatures for Real-Time Virtual Environments, *SIGGRAPH 95 Conference Proceedings*, 1995.

[Brooks, 1991] Brooks, R., Intelligence Without Reason, "Computers and Thought" IJCAI 1991.

[Downie 2001] Behavior, Animation and Music: The Music and Movement of Synthetic Characters, *Unpublished S.M. Thesis, MIT Media Lab*, 2001.

[Drescher 1991] Drescher, G.L. Made-Up Minds:A Constructivist Approach to Artificial Intelligence, MIT Press, Cambridge 1991

[Firby 1992] Building Symbolic Primitives with Continuous Control Routines. in: *Proceedings of the First International Conference on AI Planning Systems*, College Park MD, June 1992, pp 62-69.

[Funge et al. 1999] Funge, J., Tu, X., Terzopolous, D., Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters, *SIGGRAPH 99 Conference Proceedings*, 1999.

[Gallistel et al. 2000] Gallistel, C. R., & Gibbon, J. Time, rate and conditioning. Psychological Review, 107, pp 289-344.

[Kline 1999] Kline, C., Observation-based Expectation Generation and Response for Behavior-based Artificial Creatures, *Unpublished S.M. Thesis, MIT Media Lab*, 1999.

[Maes 1990] Maes, P. "Situated Agents can have Goals", Robotics and Autonomous Systems, Vol 6.

[McCallum 1995], McCallum, A. K. "Reinforcement Learning with Selective Perception and Hidden State", Ph.D. Thesis, CS Department, University of Rochester, 1995.

[Minsky 1985] Minsky, M., Society of Mind, Simon & Schuster, New York 1985.

[Perlin et al. 1996] Perlin, K., Goldberg, A., Improv: A System for Scripting Interactive Actors in Virtual Worlds, SIGGRAPH 1996.

[Reynolds 1987] Reynolds, C. W., Flocks, Herds, and Schools: A Distributed Behavioral Model, *SIGGRAPH 87 Conference Proceedings*, 1987.

[Rose et al. 1999] Rose, C.F., Cohen, M., Bodenheimer, B., Verbs and Adverbs: Multidimensional Motion Interpolation - *IEEE Computer Graphics And Applications*, Vol 18, Number 5, 1999.

[Rosenbloom et al. 1993] Rosenbloom, P.S., Laird, J.E. & Newell, A. (1993) *The Soar Papers: Readings on Integrated Intelligence.*Cambridge, MA: MIT Press.

[Treisman 1998] Treisman, A. (1998). The binding problem. In L. Squire & S. Kosslyn (Eds.), *Findings and Current Opinion in Cognitive Neuroscience*. Cambridge: MIT Press, pp 31-38, OR, Current Opinion in Neurobiology, 1996, 6, pp 171-178.

[Tu et al. 1993] Tu, X., Terzopoulos, D., Artificial Fishes: Physics, Locomotion, Perception, Behavior, SIGGRAPH 1993.

[Yoon et al. 2000] Yoon, S., Blumberg, B.M., Schneider, G.E., Motivation-Driven Learning for Interactive Synthetic Characters, *Autonomous Agents 2000 Conference Proceedings*, 2000.

Behavior Planning for a Reflexive Agent

Berardina De Carolis[3], Catherine Pelachaud[1], Isabella Poggi[2], and Fiorella de Rosis[3]

[1] Dipartimento di Informatica e Sistemistica, Università La Sapienza, Roma
[2]Dipartimento di Scienze dell'Educazione, Universita' Roma 3
[3]Dipartimento di Informatica, Università di Bari

Abstract

The aim of our research is to build a Reflexive Agent, that is able to either manifest an emotion it is feeling or to hide it. If the Agent decides to manifest its emotion, it can establish what verbal or nonverbal signals to employ in its communication and how to combine and synchronize them. In the decision of whether to express an emotion in a given context, a number of factors are considered, such as the Agent's own personality and goals, the Interlocutor's characteristics and the context. In planning how to communicate an emotion, various factors are considered as well: the available modalities (face, gaze, voice etc); the cognitive ease in producing and processing the various signals; the expressiveness of every signal in communicating specific meanings; and, finally, the appropriateness of signals to social situations.

1 Introduction

Artificial Agents are not yet endowed with the capacity of 'feeling' emotions, mainly because emotions would imply an involvement of the hardware of a machine and this will be impossible, or at least difficult, still for some years [Picard, 1997]. But let us suppose that we have an agent that is able to feel emotions; what could it do? How could it behave? Would it manifest its emotions to a potential interlocutor or would it ruminate on its own emotion by itself, without showing its feeling?

The effects of sharing an emotion, to the person who feels, are described in the psychology literature [Rimé, 1987]: it has been shown, for example, that people who have undergone severe shocks and share their emotions are better adapted, undergo fewer heart and pressure problems and have longer life [Pennebaker, 1989]. Of course, people differ in their proneness to display emotions: some of us are very impulsive and tend to display their emotions whatever the consequences of this display, while others are more reflexive and follow the rule "think it over seven times seven before". In fact, if I show to my boss that I am angry at him, this may make me feel better at the moment but I may risk being fired. In some cases, though, this may also show that I'm not afraid of him, and he might respect or admire me. Now, the decision of whether to display my anger at my boss may be a very cold, calculated and utilitarian one. But we may also decide whether to display our emotion on the spot, in a not so conscious and rational way; still, also in this case we think it is possible to speak of a "decision" of whether to display emotions, even if this decision is not deliberate and conscious. The aim of our research is to build an agent that is able to express its emotions but also to refrain from expressing them: a reflexive, not impulsive agent.

The idea is to build a behavior planner that starts from a discourse plan which represents the information content and the structure of a "non-affective" text, and enriches it so as to include the 'order' of displaying some emotion. This emotion will be finally expressed by verbal or non-verbal signals, or both. This generation process allows our system to synchronize various kinds of signals and to scatter them across the media (voice, face, gesture). For example: emphasis (or topic-comment) information may be outputted by means of a pitch accent, by raising eyebrows or with a head nod. The occurrence of all signals at the same time denotes an emphasis of the emotion to be manifested. Therefore, representing detailed information on verbal and non-verbal signals in the behaviour planner ensures a more subtle and refined multi-modal discourse generation.

After reviewing works related to ours, we present a general overview of the factors that affect the decision of whether to display an emotion. Then, we give on overview of our system by emphasizing how our discourse plan is enriched with emotion display information, and we provide an example of how our system works. We end this paper with some concluding remarks and a view on our future work.

2 Related Work

In the construction of embodied agents capable of expressive and communicative behaviors, an important step is to reproduce affective and conversational facial expressions on synthetic faces [Ball and Breese, 2000, Cassell *et al*., 1994, Cassell *et al*., 1999, Lester *et al*., 2000, Poggi *et al*., 2000, Rickel and Johnson, 1999]. For example, REA is an interactive agent that is able to converse with a user in real-time [Cassell *et al*., 1999]. REA exhibits refined interactional behaviors such as gestures for feedback or turn-taking functions. Cassell and Stone [Cassell and Stone, 1999] designed a multi-modal manager whose role is to supervise the distribution of behaviors across the several channels (verbal, head, hand, face, body and gaze). Cosmo [Lester *et al*., 2000] is a pedagogical agent particularly keen on space deixis and on emotional behavior: a mapping between pedagogical speech acts and emotional behavior is created by applying Elliott's theory [Elliott, 1992]. Ball and Breese [Ball and Breese, 2000] apply bayesian networks to link emotions and personality to (verbal and non-verbal) behaviors of their agents. André *et al.* developed a rule-based system to simulate dialogues between lifelike characters with different personality traits (extroversion and agreeableness) [André *et al*., 2000]. Marsella developed an interactive drama generator, in which the behaviors of the characters are consistent with their emotional state and individuality [Marsella, 2000].

In most of the mentioned agents, behaviors are viewed as responses to events and actions and to the way they affect emotional reactions: what is simulated is how the Agent responds with an emotion to what happens in the external context, and how an emotion affects its behavior [Marsella., 2001]. The system we propose in this paper aims at developing an Agent that is able to be affected by the emotion, but at the same time to decide whether to express it or to refrain from expressing it, not only on the basis of the emotion *per se* but also according to the context in which interaction takes place. Formalization of this reasoning considers, as we will see, several variables, such as the Agent's goals and personality and information about the Interlocutor.

3 Emotion Triggering Factors

Emotions are a biological feed-back device whose function is to provide information about the state of achievement or thwarting of our most important goals. Every time something relevant happens (or is assumed to happen) in the environment, in such a way that an important goal of the Agent (the goal of survival, of body safety, of reproduction etc) is, or is likely to be, achieved or thwarted, the feed-back device of Emotion is activated. This consists of a set of somatic, physiological, psychological, expressive and motivational issues that alert the Agent and, at the same time, provide the necessary energy and resources for reaction [Castelfranchi, 2000].

Ortony *et al.*[Ortony *et al*., 1988] and, subsequently, Elliott [Elliott, 1992] defined three entities that may be responsible for the arousal of an emotion: events, actions and objects. The occurrence of an event or some particular aspects of an object may fire appraisal of an emotion; actions of oneself or others can give rise, as well, to emotions (say, reproach or shame). Based on the Agent's interpretation of the situation, an emotion is triggered. In this work, we refer to emotion types as defined in Elliott's Affective Reasoner, in which 24 emotions are described [Elliott, 1992].

4 Emotion Regulation Factors

The expressive part of the emotional reaction (the fact, for example, that we open eyes wide in fear, or make a frown in anger, or blush in shame) usually has a precise function. Showing my rival I am angry with him may induce him to leave the field; showing my terror with wide open eyes may alert other co-specifics that a serious danger is present. This is why all theories of emotions also elaborate on the expressive aspects of the emotions and consider the expressive device as an integral part of the emotion itself [Ekman and Friesen, 1975].

Sometimes, though, expressing our emotion may be dangerous rather than useful. If I show fear in front of a rival, this may give him a weapon and may oblige me to leave the field. Humans, then, learned to be flexible in the use of the expressive part of the Emotion syndrome: that we are biologically endowed with a repertoire of display devices does not necessarily mean that any time we feel an emotion, we immediately and unthoughtfully display it. A famous elaboration around the topic of what drives the decision of whether to display an emotion is the notion of *display rules*, that is of culture-dependent rules that establish when, how and to whom to express one's own emotions. Ekman and Friesen studied how people tend to intensify, de-intensify, hide or mask their emotions according to social and cultural norms [Ekman and Friesen, 1975].

In this paper, we propose a formalization of some of these rules: we concentrate on whether one displays one's emotion or not without taking into account the subtlety of more intense or less intense display or, *a fortiori*, the masking of felt emotions (this will be the subject of our future work). We start from the following question. Suppose I am feeling a quite strong emotion, triggered by an event, an action or an object: on what basis will I decide whether to display it or not? Which are the factors that affect this decision? In our view, this depends on two aspects of the emotion: on one side, on the very nature of the emotion itself (emotional nature [Ortony *et al.*, 1988, Castelfranchi, 2000]); on the other side, on its interaction with 'scenario factors' [Poggi and Bartolucci, in prep.]. In Sect. 4.2 we overview some of these factors, while being conscious that the factors and

the associated values we are proposing may not be exhaustive, due to the complex nature of the emotional phenomena.

4.1 Emotional Nature

Emotion Valence: emotions may be either positive (when a goal is achieved, i.e., a desirable event occurs) or negative (when a goal is thwarted, an undesirable event occurs): feeling them may therefore be *pleasant* or *unpleasant*.

Emotion Social Evaluation: emotions are subject to social evaluations; for instance: showing envy is sanctioned [Castelfranchi, 2000]). They may therefore be *approved* or *sanctioned*.

Emotion Addressee: we may distinguish between social emotions (that are necessarily felt **towards** some person, as love or sense of guilt) and non-social emotions (as joy or fear). Social emotions may be addressed to the interlocutor itself (I) or towards a third person (O).

4.2 Scenario Factors

Agent's Display Motive: we call Display Motive the reason - the specific goal - that induces us to display a particular emotion in a particular situation [Poggi and Bartolucci, in prep.]. An Agent may want her Interlocutor(s) to feel the same emotion she is feeling. Here, the display motive is the goal that others feel empathy with her, that, at least, they participate to her emotion. In other cases the Agent displays her emotion because she wants the other to console her, to give her advice, to help her, to pity her or to reassure her about her feeling that emotion. Sometimes, she may just want an objectivation, in order to better understand herself. If the Agent's goal is just to give vent to her emotion, this goal does not imply another person: she wants to display it just to give vent to the physiological energy caused by the emotion, without needing another person to cause this process. There are, though, other cases where the Agent needs the presence of another person to display her emotion: I may confide a sad experience to a person in order to show her she's not more unlucky than others, or to strengthen relationships; even, I may show admiration to a person, to adulate him. A non-exhaustive list of the Display Motive factor includes the following: *vent*, *empathy*, *consolation*, *advice*, *help*, *reassurance*, *objectivation*...

Agent's Personality: an impulsive person will tend to display emotions more often than a shy or ruminative (non-impulsive) person.

Interlocutor's Features: the Agent may take into account, as well, some features she attributes to the Interlocutor. The following features may be relevant:

- **Interlocutor's Personality:** the Interlocutor's Personality interacts with the Agent's Display Motive: if I look for empathy, but I believe that you are a 'cold' and 'egocentric' person, I will probably avoid displaying my emotion to you. This factor also interacts with the specific emotion to display: I will not display my fear to someone who is apprehensive. A non-exhaustive list of these factors is the following: *envious*, *modest*, *egocentric*, *altruist*...
- **Interlocutor's Cognitive Capacity:** the factors in the following categories may take Yes/No values.
  - *Comprehension*: ability to cognitively understand the causes of the Agent's emotion.
  - *Experience*: previous experience, by the Interlocutor, of similar events that elicited the Agent's emotion. Experience interacts with the Display Motive.
  - *Problem solving*: the planning capacity of the Interlocutor to solve practical problems may be relevant (e.g. when the Agent's Display Motive is to get help or advice).

Agent-Interlocutor Role Relationship, or power relationship between people: I may avoid displaying my anger to my boss because he has power over me and can retaliate. For the sake of simplicity, we consider the following power relationships between agent (Ag) and interlocutor (I): *Ag has power over I*, *I has power over Ag*, *neutral*.

Agent-Interlocutor Personal Relationship, that is, the aggressive or friendly attitude between them: this may take four values, depending on the sense of the relationship (either Ag-to-I or I-to-Ag) and on its valence (either aggressive or adoptive). Notice that we call 'adoptive' a relationship in which one is taking care of the other's goals. The values are then: *Ag adoptive I*, *Ag aggressive I*; *I adoptive Ag*, *I aggressive Ag*.

Type of social interaction: the last important factor in the regulation of emotions is whether interaction occurs in public or not. This interacts with the Agent's Personality: the more impulsive I am, the more I'll be likely to display emotions even in public; the more shy I am, the more interaction in public will inhibit my display.

5 The Agent's Behavior Planner

In order to build the reflexive component of our Agent's mind and to interface it with its body, we developed a prototype of a Behavior Planner that is based on the architecture shown in Figure 1. We will use the following example to describe, step by step, how the system works.

"A 3 year old girl wants to give a present to her mummy; she cuts some stripes from mummy's suit and knots them together as to make a ribbon. Mummy comes in and sees the massacre".

In such a situation, the Agent (Mummy) reproaches her daughter of such an action. The input of our Discourse Generator is therefore the communicative goal: Reproach (Mummy Daughter 'cut suit'). The first step of the process consists in producing a 'discourse step of the process consists in producing a 'discourse plan' (D-Plan) to achieve this communicative goal. This plan does not yet contain any emotional content or reference (we call it a "non-affective" D-Plan); it can be produced by a planning algorithm, from a *library of plan operators* [Moore and Paris, 1993] or may be retrieved from a *library of plans* [De Carolis *et al.*, 2000].

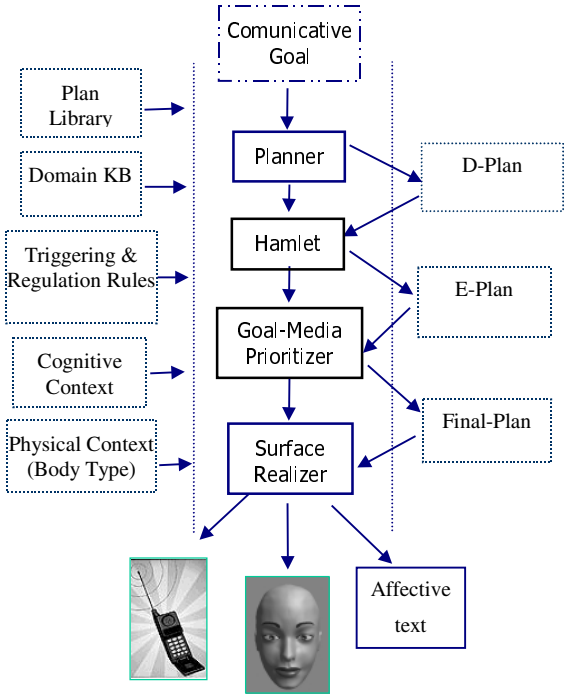


Figure 1. The Behavior Planner

In both cases, the output is a plan tree that defines the content and order of presentation of information to be conveyed in the discourse. In our case, the planner uses a library of non-instantiated plans to select the one that satisfies the posted goal. If such a plan is not available in the library, the planner builds it by appropriately combining the existing plans. The generic plan is, then, instantiated by filling the variable-slots with domain data. Part of the D-plan that is generated in our example is shown in Figure 2.

After generating the "non-affective" D-Plan, the *Hamlet* Component is activated. *Hamlet* takes as an input the *triggering rules*, that decide whether to fire the emotion; it then applies the *regulation rules* by processing the Scenario Factors, to decide whether to display the fired emotion. The triggering rules are drawn by Elliott's Affective Reasoner [Elliott, 1992], while the regulation rules are similar to the *display rules* in [Ekman and Friesen, 1975]; they are, however, enriched to encompass the Scenario Factors described above. As we anticipated in Section 4, our System does not take only cultural norms into account, but a very detailed model of the Context and of the Interlocutor. The structure of the Hamlet module is shown in Figure 3.

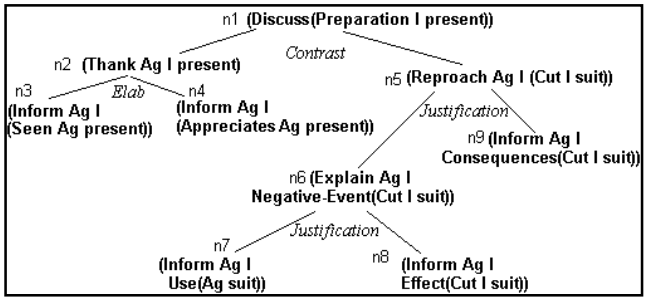


Figure 2: An example of D-Plan

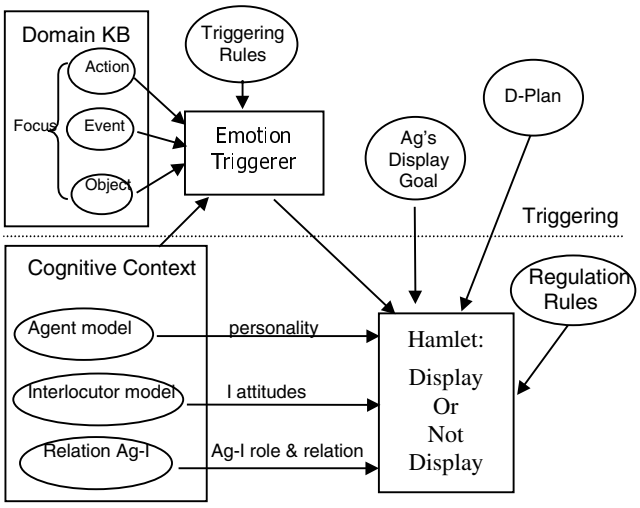


Figure 3: The Hamlet Module

Let us now see some examples of *triggering* and *regulation rules*.

The general structure of the triggering rules is the following:

```
IF DC-Cond THEN (Feel Ag e),
```

where DC-Cond represents a condition on the context or the domain, Ag denotes the agent that is conveying the signal, and e an emotion. The evaluation of DC-Cond as, for example, a Desirable or Undesirable Event or Object, is determined through a pattern-matching with the Domain KB; Feel is a predicate which applies to the two mentioned terms, to denote that 'the agent Ag feels the emotion e'. For social emotions, this predicate applies, instead, to three terms: the Agent, the emotion,

and the one to whom the emotion is addressed (the Interlocutor I or another agent O). Events, Objects and Actions are represented, in D-Plan, in the 'discourse focus' that is associated with every node.

Let, for instance, n_5 be the considered node in D-Plan; the focus of this node is (Cut I suit), which is an 'undesirable action', to the agent Ag. The triggering rule that will applied in this case is the following:

- **R1:** IF (Focus(n_i) act) AND (Undesirable act) AND(Performed I act) AND (Blames Ag I act)) THEN (Feel Ag Anger)

That is: "Ag feels anger if the focus of n_i refers to an 'undesirable' action that was performed by the interlocutor I and Ag disapproves this action.

In our example, **R1** is activated because the Domain KB mentions that the action 'cut suit' is labelled as 'Undesirable' in I is the agent who performed it.

Once an emotion has been triggered, the *regulation rules* are activated, to decide whether this emotion has to be displayed in the given context. This decision is taken by considering context features, such as: the Agent's display motive, personality and role, the Interlocutor's attitudes and the relation between the two Agents.

The general structure of the *regulation rules* is the following:

```
IF (Feel Ag e) AND DC-Cond
     THEN (Display Ag e),
```

or

```
IF (Feel Ag e) AND DC-Cond
     THEN NOT (Display Ag e).
```

As for the *triggering rules,* DC-Cond is instantiated, also in this case, by matching its variables with the content of the Domain KB. If we go back to our example, the following *regulation rules* are activated to decide whether to display the emotion:

- in the first case, Mummy is angry but she loves her daughter and knows she could not understand how distructive was the action she was doing. Therefore, Mummy will not display her anger.

  R2: IF (Feel Ag Anger) AND (Adoptive Ag I) AND NOT (UnderstandConsequences I act) THEN NOT(Display Ag Anger)

- in the second case, Mummy thinks her daughter had comprehension capacity:

  R3: IF (Feel Ag Anger) AND (Adoptive Ag I) AND (UnderstandConsequences I act) THEN (Display Ag Anger).

The enriched plan E-Plan is produced by augmenting D-Plan with the 'display' commands through application of these rules. The E-Plan corresponding to the second case is shown in Figure 4. Notice that every node inherits the 'Display' label from its parent-node, while the 'NotDisplay' label is simply not inserted in the plan: so, anger will be felt but not displayed in this case, while joy is felt and displayed, because 'present' is a 'desirable object' and there is no reason for hiding the joy emotion it triggers. On the contrary, anger would be felt *and* displayed in the second case.

In the next step, E-Plan is transformed into an XML document and is sent to the Goal-Media prioritizer. This module decides how to distribute the signals that enable displaying an emotion over the different media. For example: Anger might be displayed in one of the following ways, or, if very intense, in the three of them: verbally (through the hyperbole *"I told you a hundred times..."*); vocally (through a loud or a high pitch voice), and facially (with tense lips and an angry frown). So, this module revises the enriched plan by deciding the type of non-verbal signals to employ at every conversational step, their combination and their synchronization with verbal communication. This is done according, again, to the context and also to the type of body that has been chosen for the Agent. The XML-output of this module is the final presentation plan, where the media have been instantiated and, when necessary, synchronized or linearized. When the Goal-Media prioritizer has selected the appropriate output form (2D/3D model, audio, text...), the XML tags are interpreted and translated into parameters to drive the given output signals.

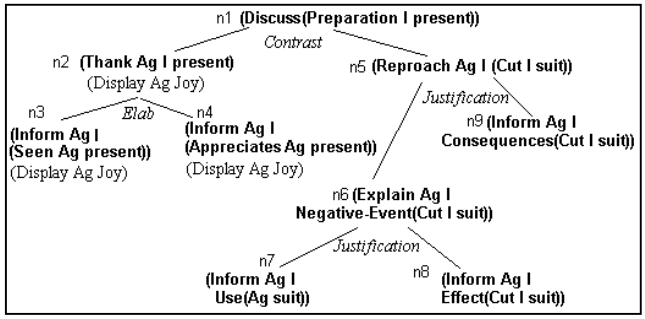


Figure 4. An example of E-Plan

At present, we are concentrating on the animation of a 3D facial model: the XML tags are converted into MPEG-4 facial parameters that drive the final animation. The XML tags are interpreted and translated into parameters to drive the given output signals; the natural language 'surface' generation step is adapted, as well, to the emotional state of the Agent. To go back to our example, in the first case, the subplan originating from node n5 will be rendered by the following sentence, pronounced with a 'not angry' face expression:
"*The ribbon you used is made from my suit, that I need for my work. If you cut it, I can't wear it anymore.*"

While in the second case, the Mummy will say, with an 'angry' face:

"The ribbon you used is made from my suit, that I need for work. I already told you a hundred times not to touch to my things!"

6 Conclusions and Future Work

In this work, we have presented the architecture of a system that builds a reflexive agent, that is, an agent who, when feeling an emotion, "decides" whether to display it immediately or not. We have also defined the different elements on which this "decision" is based (emotional nature and scenario factors). Our behaviour planner formalises these elements and then plans how to produce and synchronize the verbal and nonverbal parts of a discourse that satisfy a given communicative goal. Two sets of rules are considered: trigerring rules, that fire an emotion, and regulation rules, that make the agent 'reflexive'. In the future, we plan to evaluate how believable is our Reflexive Agent.

References

[Andre *et al*, 2000] E. Andre, T. Rist, S. van Mulken, M. Klesen, and S. Baldes. The automated design of believable dialogues for animated presentation teams. In S. Prevost J. Cassell, J. Sullivan and E. Churchill, editors, *Embodied Conversational Characters*. Cambridge, MA: MIT Press, 2000.

[Ball and Breese, 2000] G. Ball and J. Breese. Emotion and personality in a conversational agent. In S. Prevost J. Cassell, J. Sullivan and E. Churchill editors. *Embodied Conversational Characters*. Cambridge, MA: MIT Press, 2000.

[Cassell *et al*, 1994] J. Cassell, C. Pelachaud, N.I. Badler, M. Steedman, B. Achorn, T. Becket, B. Douville, S. Prevost, and M. Stone. Animated conversation: Rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents. In *Computer Graphics Proceedings, Annual Conference Series*, pages 413--420. ACM SIGGRAPH, 1994.

[Cassell *et al*, 1999] J. Cassell, J. Bickmore, M. Billinghurst, L. Campbell, K. Chang, H. Vilhjalmsson, and H. Yan. Embodiment in conversational interfaces: Rea. In *Proceedings of CHI'99*, pages 520--527, Pittsburgh, PA, 1999.

[Cassell and Stone, 1999] J. Cassell and M. Stone. Living hand and mouth. Psychological theories about speech and gestures in interactive dialogue systems. In *Proceedings of AAAI99 Fall Symposium on Psychological Models of Communication in Collaborative Systems*, 1999.

[Castelfranchi, 2000] C.Castelfranchi. Affective Appraisal versus Cognitive Evaluation in Social Emotions and Interactions. In A.Paiva editor. *Affective Interactions. Towards a New Generation of Computer Interfaces.* Berlin: Springer, 2000.

[De Carolis *et al.*, 2000] B. De Carolis., C. Pelachaud and I. Poggi. Verbal and nonverbal discourse planning. In *Proceedings of International Agents 2000 Workshop on Achieving Human-Like Behavior in Interactive Animated Agents*. Barcelona, 2000.

[Ekman and Friesen, 1975] P.Ekman and W. Friesen. *Unmasking the Face: A guide to recognizing emotions from facial clues*. Prentice-Hall, Inc., 1975.

[Elliott, 1992] C. Elliott. An Affective Reasoner: A process model of emotions in a multiagent system, Technical Report No. 32 of The Institute for the Learning Sciences Northwestern University,1992.

[Lester *et al.*, 2000] J.C. Lester, S.G. Stuart, C.B. Callaway, J.L. Voerman, and P.J.Fitzgerald. Deictic and emotive communication in animated pedagogical agents. In S. Prevost J. Cassell, J. Sullivan and E. Churchill, editors, *Embodied Conversational Characters*. MITpress, 2000.

[Mann and Thompson, 1988] W.C Mann and S. Thompson. Rhetorical Structure Theory: Towards a Functional Theory of Text Organization. *Text*, 8(3):243-281, 1988.

[Marsella, 2000] S C Marsella. Sympathy for the Agent: Controlling an Agent's nonverbal repertoire. In: Achieving Human-like Behavior in Interactive Animated Agents. 4th Int Conf Autonomous Agents, Barcelona, 2000.

[Moore and Paris, 1993] J.D. Moore and C. Paris. Planning Text for Advisory Dialogues: Capturing Intentional and Rhetorical Information. *Computational Linguistics*, 19(4), 651-694, 1993.

[Ortony *et al.*, 1988] A. Ortony, G.L. Clore and A. Collins. *The Cognitive Structure of Emotio*ns. Cambridge University Press, 1988.

[Pennebaker, 1989] J.W. Pennebaker. Confession, inhibition and disease. in L.Berkowitz (Ed.), *Advances in Experimental Social Psychology* 22. Orlando, FL.: Academic Press, 1989, 211-244.

[Picard, 1997] R. Picard. *Affective Computing*, MIT Press, 1997.

[Poggi *et al.*, 2000] I. Poggi, C. Pelachaud, and F. de Rosis. Eye communication in a conversational 3D synthetic agent. *Special Issue on Behavior Planning for Life-Like Characters and Avatars of AI Communications*, 2000.

[Poggi and Bartolucci, in prep.] I.Poggi and L.Bartolucci. Factors affecting the displaying of Emotions. In preparation.

[Rickel and Johnson, 1999] J. Rickel and W.L. Johnson. Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied Artificial Intelligence*, 13:343--382, 1999.

[Rimé, 1987] B.Rimé. Le partage social des Emotions. In B.Rimé et K. Scherer editors. *Les Emotions*. Genève: Delachaux et Niestlé, 1987.

MULTI-AGENT SYSTEMS

COOPERATIVE BEHAVIOR

Rational Competitive Analysis

Moshe Tennenholtz *
Computer Science Department
Stanford University
Stanford, CA 94305

Abstract

Much work in computer science has adopted competitive analysis as a tool for decision making under uncertainty. In this work we extend competitive analysis to the context of multi-agent systems. Unlike classical competitive analysis where the behavior of an agent's environment is taken to be arbitrary, we consider the case where an agent's environment consists of other agents. These agents will usually obey some (minimal) rationality constraints. This leads to the definition of *rational competitive analysis*. We introduce the concept of rational competitive analysis, and initiate the study of competitive analysis for multi-agent systems. We also discuss the application of rational competitive analysis to the context of bidding games, as well as to the classical one-way trading problem.

1 Introduction

Competitive analysis is a central tool for the design and analysis of algorithms and protocols for decision making under uncertainty [Borodin and El-Yaniv, 1998]. It is a well studied and widely applicable approach that fits the framework of qualitative decision-making in AI (see e.g. [Doyle and Thomason, 1997]). The competitive analysis approach attempts to minimize the ratio between the payoff an agent obtains and the payoff he could have obtained had he known the behavior of the environment. For example, consider the following trading problem (see [Borodin and El-Yaniv, 1998], Chapter 14). An agent who holds \$100 may wish to exchange them for British pounds. At each point in time, e.g. every minute in between 8AM and 4PM, an exchange ratio of dollars and pounds is announced. This ratio changes dynamically, and in an unpredicted manner. The agent would need to choose the time in which it will trade his \$100 for pounds. Notice that if the agent would have known the sequence of exchange rates, e, then he could have chosen a strategy $o(e)$ that maximizes his payoff. If the agent chooses a strategy s, then we can compute the ratio of the payoffs obtained by $o(e)$ and s. We can do similarly for every sequence of exchange rates e'. Based on this, we can compute the highest (i.e. worst) ratio, over all possible sequences, that might be obtained when we compare optimal strategies to s. This ratio is denoted by $R(s)$. According to the competitive analysis approach, the agent will apply the *competitive ratio* decision criterion: he will choose a strategy s for which $R(s)$ is minimal. This decision criterion may be quite helpful when we lack probabilistic assumptions about the environment. For example, assume that the minimal value of $R(\cdot)$, which is obtained by some strategy s, is 2. Then, by selecting s, the agent guarantees himself a payoff which is at least half of the optimal payoff that he could have obtained had he known the actual environment behavior.

The competitive ratio has also an additive variant (also termed minimax regret [Milnor, 1954]), where we replace the term "ratio" by the term "difference" in the definition of R. So, in our example, if a strategy s that minimizes $R(\cdot)$ obtains a competitive difference/regret of 20, then this implies that by performing s the agent gets a payoff (e.g. worth in British pounds) which is at most 20 lower than what he could have obtained had he known the behavior of the environment. In the sequel, we will use the additive version of the competitive ratio decision criterion.[1]

Competitive analysis has been applied to a variety of classical problems in computer science, such as the k-server problem [Koutsoupias and Papadimitriou, 1995] and paging [Fiat *et al.*, 1991], as well as to more general algorithmic problems [Borodin *et al.*, 1992]. In all of these studies the environment that the agent acts in is non-strategic, and therefore does not assume to follow any "rational" behavior. In this paper we extend the concept of competitive analysis to the context of multi-agent systems. In a multi-agent system the environment

*Permanent address: Faculty of Industrial Engineering and Management, Technion–Israel Institute of Technology, Haifa 32000, Israel

[1] In fact, as Brafman and Tennenholtz [Brafman and Tennenholtz, 2000] show the minimax regret and the competitive ratio decision criteria have the same expressive power in presenting an agent's choice among actions.

in which an agent takes his decision consists of other "rational" agents. Following previous work on competitive analysis, our approach is non-Bayesian and normative; we would like to find a decision rule for the agent that will rely as little as possible on assumptions about the behavior of his environment. Therefore, we adopt the requirement that agent A should rule out a behavior b_1 of agent B only if b_1 is dominated by another behavior, b_2, of that agent. Dominated behaviors are purely irrational in any decision making model. The agent will choose his behavior according to the competitive ratio decision rule. However, he should consider only rational behaviors of the other agents; a behavior of an agent will be considered irrational if and only if it is dominated by another behavior of it.

In section 2 we describe bidding games, a family of games that will serve us for the illustration of the basic concepts developed in this paper. Bidding games are representatives of k-price auctions, a central class of economic mechanisms [Monderer and Tennenholtz, 2000]. In section 3 we present a competitive analysis of bidding games. In section 4 we introduce rational competitive analysis, a new tool for normative decision making, that generalizes competitive analysis to the context of rational environments, and apply it to bidding games. In section 5 we consider repeated (multi-stage) games. We present several results on the relationships between (rational) competitive analysis of repeated games and the competitive analysis of the particular (one-shot) games they consist of. Then, in section 6, we discuss and study variants and modifications of the one-way trading problem, using rational competitive analysis. In particular, we study the multi-agent one-way trading problem.

2 Bidding games

We start by recalling the general definition of a (strategic form) game.

Definition 2.1 *A* game *is a tuple* $G = \langle N = \{1, 2, \ldots, n\}, S = \{S_i\}_{i=1}^n, \{U_i\}_{i=1}^n \rangle$, *where* N *is a set of* $n \geq 2$ *players,* S_i *is the set of strategies available to player* i, *and* $U_i : \Pi_{j=1}^n S_j \rightarrow R$ *is the utility function of player* i.

In a game, each player selects a strategy from a set of available strategies. The tuple of strategies selected, one by each player, determines the payoff of each of the agents[2] (as prescribed by the utility functions).

In a bidding game, a center attempts to obtain a service from a set of potential suppliers. Each such supplier has a certain cost associated with that service. This cost is taken to be an integer between $K - T$ and K, where K and T are w.l.o.g integers, $K > T > 0$. Each agent will offer his service and ask for a payment in the range in between $K - T$ and K. We will associate the request for payment of $K - i$ with the integer i, where $0 \leq i \leq T$. The center will choose as the service provider the supplier with the lowest asking price. There are various ways for determining the payment to that agent; in particular, the agent can be paid his asking price, the second lowest asking price, or the third lowest asking price. We assume that the costs for providing the service by each of the agents are common knowledge among them, although the center might not be familiar with these costs. Although this is quite natural for the above procurement problem, other assumptions can be treated similarly.

Our definition of bidding games will capture the above, by considering a fully isomorphic situation, namely: the auctioning of a good. The good is held by the center. Each agent has a valuation (i.e. maximal willingness to pay) for the good. Each agent needs to decide on his bid. The center will allocate the good to the agent with the highest bid (rather than to the agent with the lowest asking price, as in the isomorphic procurement problem).

Formally, we have:

Definition 2.2 *Given a set of* n *players, and an integer* $T >> 1$, *a bidding game is determined by the tuple* $B=(x_1, \ldots, x_n, k)$ *where* $x_i = \frac{l_i}{T}$ *for some integer* $0 \leq l_i \leq T$, *and* $1 \leq k \leq n$ *is an integer. Player* i*'s strategy in* B *is a decision about* $b_i \in [0, T]$. *Given a strategy profile* $b = (b_1, b_2, \ldots, b_n)$ *denote by* $b_{[j]}$ *the* j*-th order statistic of this tuple. Let* $M(b)$ *be the number of elements of* b *that equals* $b_{[1]}$. *Then,* $U_i(b) = \frac{1}{M(b)}(x_i - \frac{b_{[k]}}{T})$ *if* $b_i = b_{[1]}$, *and* $U_i(b) = 0$ *otherwise.*

In the above formalism, x_i is the valuation of agent i (that is normalized to the interval [0,1]), while b_i denotes the bid made by agent i. The payment made by the winner is determined by the parameter k. If $k = 1$ we get the standard high-bid wins (or first-price) auction; if $k = 2$ then we get the famous Vickrey (second-price) auction, while if $k = 3$ we get the case of third-price auctions.[3]

For ease of presentation we will assume that $2 \leq l_i < T$ for every $1 \leq i \leq n$, that $i \neq j$ implies $l_i \neq l_j$, and that $T \geq n$.

3 Competitive Analysis

In a game, agent i is facing an environment that consists of the other agents. The actions to be selected by these agents are not under the control of i. Following the literature on competitive analysis, the competitive ratio decision rule may be used in order to choose an action for that agent.

Definition 3.1 *Given a game* G, *and a strategy profile* $s = (s_1, s_2, \ldots, s_n) \in \Pi_{j=1}^n S_j$, *the regret of player* i *is given by* $Reg_i(s_i, s_{-i}) = max_{t \in S_i} U_i(s_1, \ldots, s_{i-1}, t, s_{i+1}, \ldots, s_n) - U_i(s)$. *A strategy* $s \in S_i$ *is a competitive strategy for agent* i *if* $s \in argmin_{t \in S_i} max_{q \in S_{-i}} Reg_i(t, q)$, *where* S_{-i} *denotes the possible strategy profiles of players in* $N \setminus \{i\}$.

[2] Here and elsewhere we use the terms player and agent interchangeably.

[3] Third-price auctions have been shown to have appealing properties in the context of Internet Auctions [Monderer and Tennenholtz, 2000].

Given the above definition we are interested in applying competitive analysis to bidding games. We now present three claims about competitive analysis of bidding games. These claims are associated with the competitive analysis of 1st,2nd, and 3rd-price auctions, respectively.

Claim 3.1 *Given the bidding game $B=(x_1,\ldots,x_n,1)$, a competitive strategy for agent i yields a regret value of $\frac{\alpha}{T}$, where α equals the upper integer value of $\frac{l_i-1}{2}$.*

Basic idea behind proof: Agent i can lose by submitting a bid that is higher than his valuation. On the other hand, by submitting a bid that is below l_i-1 agent i might lose, since agent $j\neq i$ might submit l_i-1 as a winning bid. Since agents may submit the bid 0, agent i will minimize his regret by submitting a bid that equals (the upper integer value of) half of the difference between l_i-1 and 0.

Claim 3.2 *Given the bidding game $B=(x_1,\ldots,x_n,2)$ a competitive strategy for agent i yields a zero regret.*

Basic idea behind proof: Here the optimal strategy for an agent, regardless of what the others do, is to send his actual valuation as his bid; this is a well known property of the Vickrey auction [Wolfstetter, 1996]. As a result we get a regret of 0.

Claim 3.3 *Given the bidding game $B=(x_1,\ldots,x_n,3)$, and assume w.l.o.g that $x_1>x_2>\cdots>x_n$, then agent j's competitive strategy is to send the bid $min(2l_j,T)$.*

Basic idea behind proof: Given that agents may submit the bid 0, agent j might reach a regret of $\frac{l_j}{T}$ if he is not the winner. Submitting however a bid that is higher than $2l_j$ may also lead to a regret of $\frac{l_j}{T}$, given that the agents may submit $2l_j$ as their bids. Combining these observations, we get that submitting the bid $min(2l_j,T)$ is the competitive strategy.

4 Rational Competitive Analysis

Although competitive analysis is a most powerful concept from a non-Bayesian normative perspective, it may be quite restrictive when we consider decision-making in multi-agent systems. Following the spirit of competitive analysis for normative decision making, we refrain from using probabilistic assumptions and game-theoretic equilibrium analysis.[4] However, one can still improve on the use of competitive analysis by considering minimal rationality requirements.

Definition 4.1 *Given a game $G=\langle N=\{1,2,\ldots,n\},\{S_i\}_{i=1}^n,\{U_i\}_{i=1}^n\rangle$, we say that a strategy $s_i\in S_i$ weakly dominates a strategy $s_i^{'}\in S_i$ if $U_i(s_i,t)\geq U_i(s_i^{'},t)$ for every strategy profile t of the players in $N\setminus\{i\}$, and there exists such strategy profile t' for which $U_i(s_i,t')>U_i(s_i^{'},t')$. A strategy $s\in S_i$ will be called rational if there is no other strategy $\bar{s}\in S_i$ that weakly dominates it. Given a game G, the set of rational strategies for player i will be denoted by $Rat(S_i)$.*

In any reasonable model agents will choose only from the set of non-dominated strategies. Our idea is therefore to combine the powerful idea of competitive analysis and this minimal requirement of rationality, in order to re-introduce competitive analysis into the framework of multi-agent systems.

Definition 4.2 *A strategy $s\in S_i$ is a rationally competitive strategy if $s\in argmin_{t\in S_i}max_{q\in Rat(S_{-i})}Reg_i(t,q)$, where $Rat(S_{-i})$ denotes the possible rational strategy profiles of players in $N\setminus\{i\}$, i.e. each player $j\in N\setminus\{i\}$ chooses its strategy from $Rat(S_j)$.*

Basically, a rationally competitive strategy applies the competitive ratio decision criterion, while taking into account only rational activities of the environment. As the following claims illustrate, rational competitive analysis introduces an improved approach to normative decision making.

Claim 4.1 *Given the bidding game $B=(x_1,\ldots,x_n,1)$, a rational competitive strategy for agent i yields a regret of $\frac{\alpha}{T}$, where α equals the upper integer value of $\frac{min(l_i,max_{j;j\neq i}l_j)-2}{2}$.*

Basic idea behind proof: We observe that any strategy that tells agent j to submit a bid which is greater than or equals to his valuation is dominated by the strategy of submitting his valuation minus 1. Given our assumptions about the possible valuations, all other strategies, excluding the strategy of submitting the bid 0, are not dominated. As a result, from the perspective of agent i, if his valuation is the highest one, he will minimize his regret if he will make a bid that is half of the distance between $max_{j;j\neq i}l_j-1$ and 1. If agent i's valuation is not the highest one then he will minimize his regret (again, taking into account the assumptions on possible valuations) if he will make a bid in between l_i-1 and 1.

Notice that rational competitive analysis allows us to improve upon the type of reasoning carried out in claim 3.1. Technically, in the case of a bidding game with $k=1$, rationality implies that we need to take the minimum between l_i and the highest other agents' valuation in our analysis, rather than consider l_i only.

Claim 4.2 *Given the bidding game $B=(x_1,\ldots,x_n,2)$ a rational competitive strategy for agent i yields a zero regret.*

As we can see, unlike the major effect of the rationality assumption in the case of a first-price auction, there is no change in the analysis in the case of a second-price auction. In the case of a third-price auction, we see again the effect of the revised notion:

Claim 4.3 *Given the bidding game $B=(x_1,\ldots,x_n,3)$, and assume w.l.o.g that $x_1>x_2>\cdots>x_n$, then a rational competitive strategy for agent j $(j=1,2)$, is to*

[4] The debate about whether competitive ratio and non-Bayesian decision making are expressive or useful for normative or descriptive objectives is beyond the scope of this paper; see [Brafman and Tennenholtz, 2000] for sound and complete axiomatization of the competitive ratio, minimax regret, and maximin decision criteria.

submit the bid $min(2l_j - l_{[3]}, T)$*, where* $l_{[3]}$ *corresponds to the 3rd highest* x_k*; a zero-regret rational competitive strategy for agent* $i, 3 \leq i \leq n$*, is to submit* l_i*.*

Basic idea behind proof: First, observe that any strategy where the agent submits a bid that is below that agent's valuation is dominated by the strategy that tells him to submit his actual valuation as his bid. As a result, for agents $3, 4, \ldots, n$ there is a 0 regret in submitting their actual valuations as their bids. Let us assume that agent i (where i is 1 or 2) submits a bid, then it can lose $\frac{l_i - l_{[3]}}{T}$ if it turns out not be the highest bidder (since agent j submits a higher bid). On the other hand, by submitting the bid $b_i > l_i$ a loss of $\frac{b_i - l_i}{T}$ may be caused, since (from the perspective of agent i) two other agents may submit the bid b_i. This implies that the bid $min(2l_i - l_{[3]}, T)$ will minimize this agent's regret.

As we can see, in the case of $k = 3$ as well, rational competitive analysis for bidding games leads to an improved normative approach to decision making. In particular, the competitive strategy of Claim 3.3 specifies a too high bid, and is not a rationally competitive strategy; as a result, it fails to serve in a multi-agent context.

5 Rational competitive analysis in repeated games: folk theorems

We first recall the notion of finitely repeated games [Fudenberg and Tirole, 1991].

Definition 5.1 *Given an integer* $l > 0$ *and a game* $G = \langle N = \{1, 2, \ldots, n\}, S = \{S_i\}_{i=1}^n, \{U_i\}_{i=1}^n \rangle$*, a* repeated game $RG = (G, l)$ *with respect to* G *is a game where* G *is repeatedly played* l *times. RG consists of the following strategies and utility functions: a strategy of agent* i *in RG determines the strategy of* G *to be taken by* i *in the* k*-th iteration of* G*, as a function of the history of strategies of* G *selected by the others in iterations* $1, 2, \ldots, k-1$*. Given a tuple of strategies of RG, one for each agent, the payoff for agent* i *is the sum of its payoffs along the* l *iterations. A sub-game of a repeated game RG is a repeated game that starts from iteration* $1 \leq q \leq l$ *of RG and consists of* $l - q + 1$ *iterations. A (rationally) competitive strategy in RG is a strategy that is a (rationally) competitive strategy at each of the sub-games of RG.*[5]

Repeated games have been of much interest in the game-theory literature, due to the fact they enable to study agents' actions as a function of past events and other agents' actions. The study of repeated games is central to the understanding of basic issues in coordination and cooperation (e.g. [Axelrod, 1984]), as well as for the study of learning in games (e.g. [Fudenberg and Levine, 1998]).

One of the central challenges for the study of repeated games is to establish general theorems (titled folk-theorems) that explain/recommend behavior in these (repeated) games by means of solution concepts for the games they consist of. In our case, it would be of interest to understand what will be a rationally competitive strategy in a repeated game, and try to relate it to the competitive analysis of the simple one-shot game that takes place at each iteration.

We now present a general result about competitive analysis in repeated games. For ease of presentation we will assume that G is a two-player game, where all payoffs are distinct. We will also assume w.l.o.g that all payoffs are non-negative. Given a repeated game (G, l), let us denote the highest payoff for agent i in G by $h_i(G)$, and the second highest payoff of agent i in G by $sh_i(G)$.

Theorem 5.1 *Given a repeated game* (G, l) *and assume that for each agent* i $h_i(G) \geq 2 \cdot sh_i(G)$*, then a rationally competitive strategy for agent* i *in the game* (G, l) *is obtained by performing a competitive strategy of it in* G *on iterations* $1, 2, \ldots, l-1$ *and performing a rational competitive strategy of it in* G *on the last iteration.*

Basic idea behind proof: From the perspective of agent i, assuming we are at stage $k < l$, the selection of any strategy s of G by j can be complemented to a non-dominated strategy of j; this non-dominated strategy will tell j to choose the strategy associated with $h_j(G)$ in stages $k+1, \ldots, l$. The reason that the resulting strategy is not dominated is that j considers the strategy where i will also choose in stages $k+1, \ldots, l$ the strategy (of his) in G that corresponds to $h_j(G)$, and does it only if in stage k agent j chooses s; in addition, according to this strategy i will choose the strategy that corresponds to $sh_j(G)$ is stage k. Therefore, given that $h_j(G) \geq 2 \cdot sh_j(G)$, we get that the selection of the strategy s in stage k can be complemented to a non-dominated strategy. This implies that agent i should consider at stages $1, 2, \ldots, l-1$ all possible strategies of agent j in G. In the last stage agent i is no longer subject to the above considerations and will choose the rationally competitive strategy of G.

The above theorem shows a strong connection between competitive analysis in repeated games and competitive analysis in simple single-shot games. As it turns out, this connection can be further generalized to a much richer context:

Definition 5.2 *Let* $\bar{G} = (G_1, G_2, \ldots, G_m)$ *be a sequence of games where* N *is the set of players in each of the games in the sequence, and game* G_i *is played in iteration* i*. The strategy of agent* t *in* $\bar{G}$ *determines its strategy in* G_i*,* $1 \leq i \leq m$*, as a function of the strategies of* G_j*,* $1 \leq j < i$*, selected by the other agents in previous iterations. Given a tuple of strategies of* $\bar{G}$*, one for each agent, the payoff of agent* i *is taken as the sum of its payoffs in the* m *iterations.*

Theorem 5.2 *Given a sequence of games* $\bar{G} = (G_1, G_2, \ldots, G_m)$ *where* N *is the set of players in each of the games in the sequence, and game* G_j *is played in iteration* j*, and assume that* $h_i(G_k) \geq 2 \cdot sh_i(G_l)$ *for every* $1 \leq k, l \leq m$*, and for every agent* i*, then a rationally competitive strategy for agent* i *in the game* $\bar{G}$

[5] This definition is in the spirit of sub-game perfect equilibrium in game theory.

is obtained by performing the competitive strategy of G_j in iterations $1, 2, \ldots, l-1$ and performing the rational competitive strategy of G_m in the last iteration.

The above theorem can be generalized into a situation where n games from among the set of games $\{G_1, G_2, \ldots, G_m\}$ are executed in some random order (with possible repetitions):

Definition 5.3 *Given a set of games $G = \{G_1, G_2, \ldots, G_m\}$, a random game with respect to G, $\bar{G}$, is a sequence of n games $(g_1, g_2, \ldots, g_n)$, where $g_i \in G$ $(1 \leq i \leq n)$ and N is the set of players in each of the games in the sequence. The game to be played in iteration i, g_i, is randomly selected from the set G independently of previous selections made. The strategy of agent t in $\bar{G}$ determines its strategy in g_i, $1 \leq i \leq n$, as a function of the strategies of g_j, $1 \leq j < i$, selected by the other agents in previous iterations. Given a tuple of strategies of $\bar{G}$, one for each agent, the payoff of agent t is taken as the sum of its payoffs in the n iterations. A sub-game of a random game $\bar{G}$ with respect to G, is a random game with respect to G that starts from iteration $1 \leq j \leq n$ and consists of $n-j+1$ iterations as above. A (rationally) competitive strategy in a random game is required to be a (rationally) competitive strategy at each sub-game of it.*

Theorem 5.3 *Given a random game $\bar{G}$ with respect to $G = \{G_1, G_2, \ldots, G_m\}$, and assume that $h_i(G_k) \geq 2 \cdot sh_i(G_l)$ for every $1 \leq k, l \leq m$, and for every agent i, then a rationally competitive strategy for agent i in the game $\bar{G}$ is obtained by performing the competitive strategy of game g_i in iterations $1, 2, \ldots, n-1$ and performing the rational competitive strategy of the game g_n on the last iteration in the sequence.*

6 One-way trading in multi-agent systems

In the previous section we have discussed competitive analysis for multi-agent systems in the framework of general repeated games and random games. In this section we look at a particular variant of repeated games that extends a well known and fundamental framework for competitive analysis – the one-way trading (see citations in chapter 14 of [Borodin and El-Yaniv, 1998]).

One way to present the structure of one-way trading is as follows. An agent a seeks buying X units of a good or of a service. A supplier A wishes to supply these units of good to a. The agents act in an environment that determines the actual payment for a unit of good in a non-deterministic way. For example, the payments might be specified in dollars, but since agent A is a British company the actual payoffs it will obtain for providing the good will depend on the exchange ratio of the dollar and the British pound. Formally, the environment announces at each point in time, $1, 2, \ldots, t$, the payoff that will be obtained by agent A for supplying a unit of good. The announcements are selected in an unpredicted non-deterministic manner from the interval $[m, M]$, where $M > m > 0$. For example, when K is announced at point i, agent A can supply the X units of good and obtain a payoff of $X \cdot K$. Our assumption is that agent A will obtain a zero payoff by not providing the units of good. The decision problem that agent A faces is as follows: at each point he needs to decide whether he would like to supply the units of good in the current rate. We assume that when agent A is willing to provide the service then he will provide and be paid for the whole quantity of goods requested by agent a (this property is termed one-way search).

The competitive analysis approach tells agent A in the above scenario to minimize his regret value. As it turns out, the competitive strategy in this case will tell the agent to accept the offer (i.e supply the units of good) when the payoff reaches $\frac{M-m}{2}$ in stage $j \leq t-1$, and to accept the offer on stage t otherwise.

One-way trading is a typical setting for the use of competitive analysis. We now extend it to the case of several agents, where more than one agent may wish to supply the units of good requested by a. We will first develop the multi-agent framework without considering the rationality assumption, and then will extend it to the case of rational competitive analysis.

6.1 Multi-agent one-way trading

For ease of exposition we consider the case of two agents (i.e. two suppliers who can provide the units of good requested by a): A_1 and A_2. The payment offers for the two agents are taken to be independent. For example, agent 1 may be a British company and agent 2 may be a Japanese company, and therefore the actual payment offers for them (from their perspective) will reflect the exchange ratio between the dollar and the British pound, and the exchange ratio between the dollar and the Japanese Yen, respectively. Formally, we have:

Definition 6.1 *Let $M_1, M_2, m_1, m_2, t, X = 2K$ be positive integers, where $M_1 > m_1$ and $M_2 > m_2$, $t \geq 3$, and X is even. A multi-agent one way trading $T = \langle N = \{1, 2\}, X, t, m_1, M_1, m_2, M_2 \rangle$ is a random game with the following players, strategies, and payoffs:*

1. *The players are 1 and 2.*
2. *There are t iterations. Each iteration i is associated with a pair of numbers (a_1, a_2) where $m_1 \leq a_1 \leq M_1$ and $m_2 \leq a_2 \leq M_2$. At each iteration each agent can "take" or "pass". However, if an agent takes in iteration j then both agents can only "pass" in all iterations $j \geq i$.*
3. *The payoff of each agent in iteration i is 0 if it passes; if an agent performs "take" in iteration i then its payoff will be $a_i X$ if the other agent passes and $a_i K$ if the other agent takes.*

Intuitively, "take" means a decision of accepting the offer, while "pass" means rejecting it (at the given point). If both agents agree to "take" then each one of them will supply half of the units (and the payoff will be splitted among the agents). We now show what is the structure of

the competitive strategy in a multi-agent one-way trading setting.

Theorem 6.1 *Given a multi-agent one way trading* $T = \langle N = \{1,2\}, X, t, m_1, M_1, m_2, M_2\rangle$, *a competitive strategy for agent* i *is as follows:*

1. *For iterations* $1 \leq j \leq t-1$, *take iff* $a_i \geq \frac{2M_i+m_i}{4}$
2. *If you arrive at iteration* t *then take.*

Basic idea behind proof: Consider iteration j, $1 \leq j \leq t-1$, and consider the announcement $a_i = Y$. Then, by taking in round j, agent i might suffer a regret of $2M_iK-2YK$ (notice that there is a regret when an agent takes only if the other does not take at that iteration). By not taking in stage j agent i might suffer a regret of $2YK - m_iK$ (which is in fact $max(KY, 2YK - m_iK)$). In order to minimize the regret agent i will therefore have to take whenever Y satisfies that $2M_iK - 2YK = 2YK - m_iK$, i.e. when $a_i = \frac{2M_i+m_i}{4}$. The fact that the regret is minimized in iteration t by taking rather than passing is immediate.

6.2 Rational competitive analysis for multi-agent one way trading

We now show the result of applying rational competitive analysis to the context of multi-agent one way trading:

Theorem 6.2 *Given a multi-agent one way trading* $T = \langle N = \{1,2\}, X, t, m_1, M_1, m_2, M_2\rangle$, *a rational competitive strategy for agent* i *is as follows:*

1. *For iterations* $1 \leq j \leq t-1$, *if the other agent,* k, *is announced that* $a_k = M_k$, *then take.*
2. *For iterations* $1 \leq j \leq t-2$, *if (1) does not hold then take iff* $a_i \geq \frac{2M_i+m_i}{4}$
3. *If (1) does not hold, then in iteration* $t-1$ *take iff* $a_i \geq \frac{M_i+m_i}{4}$.
4. *If you arrive at iteration* t *then take.*

Basic idea behind proof: Notice that if the other agent, k, is announced that $a_k = M_k$ then taking dominates any other strategy of it. In no other cases we can say that taking or passing in iterations $1, 2, \ldots, t-1$ is dominated. Also, in stage t passing is dominated by taking. As a result we will get that agent i will minimize its regret by taking when $a_k = M_k$ or when it arrived in the last iteration. Assume that $a_i = Y$ in iteration $t-1$, then the maximal regret we get by taking is $M_iK - 2KY$, and by passing the maximal regret in this case is $2KY - m_iK$ (which is in fact $max(KY, 2KY - m_iK)$). This implies that the regret is minimized when $M_iK - 2YK = 2KY - m_iK$), i.e. when $Y = \frac{M_i+m_i}{4}$. The case that refers to iterations $1, 2, \ldots, t-2$ is treated as in Theorem 6.1.

7 Conclusion

Competitive analysis is a major tool in computer science, which has been used in a variety of contexts. In this paper we have introduced rational competitive analysis. Rational competitive analysis generalizes competitive analysis to the context of multi-agent systems. Moreover, we have shown its use in the context of bidding games and one-way trading, two problems of considerable importance, as well as in the context of general repeated games. Our approach adopts the non-Bayesian normative approach adopted in previous work, but modifies it to incorporate minimal rationality requirements. Such requirements are essential in multi-agent domains. Many of the previous studies in the context of competitive analysis can be naturally extended to multi-agent domains, and then rational competitive analysis can serve as a fundamental tool for the study of these extensions. We see the study of such extensions as a most attractive research topic, and hope that others will join us in addressing this challenge.

References

[Axelrod, 1984] R. Axelrod. *The Evolution of Cooperation.* New York: Basic Books, 1984.

[Borodin and El-Yaniv, 1998] Allan Borodin and Ran El-Yaniv. *On-Line Computation and Competitive Analysis.* Cambridge University Press, 1998.

[Borodin *et al.*, 1992] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. *Journal of the ACM*, 39:745–763, 1992.

[Brafman and Tennenholtz, 2000] R. I. Brafman and M. Tennenholtz. An axiomatic treatment of three qualitative decision criteria. *Journal of the ACM*, 47(3), March 2000.

[Doyle and Thomason, 1997] J. Doyle and R. Thomason. Qualitative preferences in deliberation and practical reasoning. Working notes of the AAAI spring symposium, 1997.

[Fiat *et al.*, 1991] A. Fiat, R.m. Karp, M. Luby, L.A. McGeoch, D.D. Sleator, and N.E. Young. On competitive algorithms paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[Fudenberg and Levine, 1998] D. Fudenberg and D. Levine. *The theory of learning in games.* MIT Press, 1998.

[Fudenberg and Tirole, 1991] D. Fudenberg and J. Tirole. *Game Theory.* MIT Press, 1991.

[Koutsoupias and Papadimitriou, 1995] E. Koutsoupias and C. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

[Milnor, 1954] J. Milnor. Games Against Nature. In R. M. Thrall, C.H. Coombs, and R.L. Davis, editors, *Decision Processes.* John Wiley & Sons, 1954.

[Monderer and Tennenholtz, 2000] D. Monderer and M. Tennenholtz. K-price auctions. *Games and Economic Behavior*, 31:220–244, 2000.

[Wolfstetter, 1996] E. Wolfstetter. Auctions: An introduction. *Journal of Economic Surveys*, 10(4):367–420, 1996.

Learning Procedural Knowledge to Better Coordinate*

Andrew Garland and **Richard Alterman**
Volen Center for Complex Systems
Brandeis University
Waltham, MA 02254
{aeg,alterman}@cs.brandeis.edu

Abstract

A fundamental difficulty faced by groups of agents that work together is how to efficiently coordinate their efforts. This paper presents techniques that allow heterogeneous agents to more efficiently solve coordination problems by acquiring procedural knowledge. In particular, each agent autonomously learns *coordinated procedures* that reflect her contributions towards successful past joint behavior. Empirical results validate the significant benefits of coordinated procedures.

1 Introduction

Research on groups of agents that work together is a large and growing field that covers topics such as autonomous robots, software agents, and smart objects. A fundamental difficulty faced by such agents is how to coordinate their efforts when they have overlapping objectives. This coordination problem is both ubiquitous and challenging, especially in environments where agents have limited knowledge about, and control over, other agents and the world around them.

This work is part of a line of research interested in groups of agents that interleave planning and execution [desJardins *et al.*, 1999; Grosz and Sidner, 1990; Levesque *et al.*, 1990; Durfee and Lesser, 1991; Decker and Lesser, 1992; Grosz and Kraus, 1996; Tambe, 1997]. In the present work, individual agents are motivated by personal objectives and do not reason about group-wide objectives or attempt to establish or maintain group-wide mental attitudes. The extent to which their activity can be successful depends on the degree to which the individuals' objectives converge.

One approach to the coordination problem is to design agents to have common built-in knowledge about the group, such as knowledge of the planning or execution abilities of all agents. This common knowledge makes agents more likely to be able to efficiently solve coordination problems that occur at runtime. Unfortunately, for many domains there will continue to be coordination problems that lie outside the initial design because of the difficulty of foreseeing all possible interactions in complex, dynamic environments.

*This work was supported in part by the ONR (grants N00014-96-1-0440 and N00014-97-1-0604).

In this paper, an agent acquires knowledge about the environment and other agents from experience, supplementing any *a priori* common knowledge she might have. Thus, individual agents learn to better coordinate their actions so that the agents' future behavior more accurately reflects what works in practice.

The learning techniques are memory-based and a novel contribution is a technique to learn *coordinated procedures* based on past, possibly unplanned, successful joint behavior. These procedures are extracts of execution traces, which are the result of multiple planning sessions occurring at various times during the activity, and are composed around, and organized by, *coordination points* [Alterman and Garland, 2001]. Unexpected requests and responses allow an agent to acquire coordination knowledge about other agents, and constitute the building blocks of learned coordinated procedures.

This paper begins by outlining the framework within which the coordination problem is studied. The next section presents the key technical details that allow agents to learn coordinated procedures. Empirical results then demonstrate that coordinated procedures provide a statistically significant improvement in run-time performance and are used efficiently when planning. The paper ends with a discussion of related work.

2 Coordinating independent agents

This section describes the aspects of the system framework that are relevant to studying the coordination problem. The most noteworthy features are the autonomy of the agents, the distribution of both problem-solving knowledge and execution ability, and the role of communication as a coordination mechanism. Given these attributes, even seemingly simple problems create imposing hurdles to efficient coordination.

2.1 An example of a coordination problem

In the test-bed domain, called MOVERS-WORLD, the task is to move boxes from a house onto a truck or *vice versa*. MOVERS-WORLD has multiple agents of different types: some are "lifters" and some are "hand-truck operators". The agents do not know their type or even have an internal representation of the concept of type. Most actions are type-specific, but all agents are able to move and communicate. The duration of conversations between two agents varies

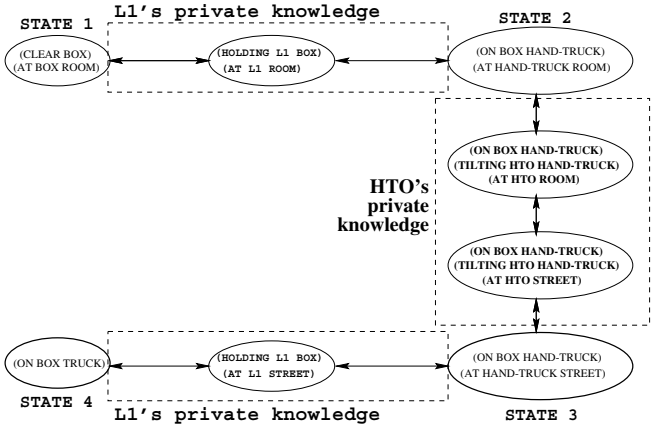


Figure 1: The distribution of problem-solving knowledge.

based on the content of the dialog. The agents have no built-in planning knowledge about the execution abilities of agents of other types. Each agent has the autonomy to decide which goal(s) to work on at any time.

The rest of this subsection discusses how coordination problems arises in MOVERS-WORLD. The plans a lifter can generate and the plans a hand-truck operator can generate are contrasted in the context of moving a single box onto a truck. By expanding the problem to include more boxes, the difference in planning ability makes achieving optimal coordination impossible and achieving even near-optimal behavior unlikely.

A solution path for a simple situation where a lifter (L1) and a hand-truck operator (HTO) could work together to get a box onto the truck is shown in Figure 1. Over the course of the solution, the box "moves" through eight different states (represented by ovals listing the salient predicates). Only L1 knows how to transform the box from state 1 to state 2. Only HTO knows how to get from state 2 to state 3. And only L1 knows how to get the box from state 3 to state 4.

L1 cannot generate a plan to match this solution path on her own since L1 has no planning knowledge of the hand-truck or of HTO's capabilities. Backward chaining can identify state 3 as a precursor to state 4 and forward-chaining can identify state 2 as a successor to state 1. However, L1 cannot distinguish the pair of states (state 2, state 3) that is relevant to this solution path from the many other pairs of states that are not relevant to any solution path.

Coordination becomes more of an issue when the agents want to move two boxes onto the truck in as little time as possible. Optimal performance involves no communication whatsoever and would take 244 ticks of a simulated clock. However, without communication, L1 would never construct a plan to load BOX1 onto the hand-truck! L1's expectation at the outset would be to move both boxes to the truck by carrying them; HTO's expectation is that both boxes would be moved via the hand-truck. Furthermore, neither knows that the other is working on the same two goals.

```
Ticks   1 to  15: HTO and L1 converse
  "L1, would you help me achieve (ON BOX1 HANDTR)?"
  "HTO, I'll help, but you'll have to wait a bit."
Ticks  16 to  45: <LIFT L1 BOX1>
Ticks  46 to  80: <LOAD L1 BOX1 HANDTR>
Ticks  81 to 100: <TILT-HANDTR HTO HANDTR>
Ticks  81 to 112: <LIFT L1 BOX2>
Ticks 101 to 125: <PUSH-HANDTR HTO HANDTR STREET>
Ticks 113 to 152: <CARRY L1 BOX2 STREET>
Ticks 126 to 145: <STAND-HANDTR HTO HANDTR>
Tick         146: HTO trying to contact L1 ...
Ticks 153 to 167: HTO and L1 converse
  "L1, would you help me achieve (ON BOX1 TRUCK)?"
  "HTO, I'll help, but you'll have to wait a bit."
Ticks 168 to 202: <LOAD L1 BOX2 TRUCK>
Ticks 203 to 237: <UNLOAD L1 BOX1 HANDTR>
Ticks 238 to 272: <LOAD L1 BOX1 TRUCK>
```

Figure 2: A near-optimal solution to a coordination problem.

The closest to optimal that is possible given baseline coordination abilities is 272 ticks as shown in Figure 2. That solution requires that L1 agrees to help HTO on both occasions and that L1 correctly adapts her plan. While none of these is unlikely independently, neither agent possesses enough knowledge to reliably act in this way. As the complexity of the problems increases by including other agents and increasing the number of boxes, there is an exponential increase in the number of decisions that all must be made "correctly" for the community to perform even this close to optimal.

Coordination problems in this domain are not solely a consequence of the distribution of planning knowledge. Even if both agents had common goals and planning knowledge, there is ambiguity about the order on which to work on the goals. Also, when there is uncertainty about the outcome of actions, they may have different preferences among alternative solutions.[1] Finally, agents may have different beliefs about the current state of the world, leading to different beliefs about the best course of action.

2.2 Communication as a coordination mechanism

In terms of the coordination problems faced by the agents, a central feature of this system is that communication, cooperation, and coordination are shaped by the autonomy of the agents. Agents do not communicate at planning time; they plan independently, act independently, and only communicate when necessary to establish cooperation or to maintain coordination. Each problem includes some goal(s) that can only be solved by agents that work together, so communication is an essential part of the community activity.

Communication happens at *coordination points*, which are defined as points in the activity where an actor cannot progress without the assistance of someone else. If all actions are reversible and the goals of the agents do not conflict, conversing at individual coordination points at the time when they arise is sufficient to ensure that the activity will be completed.

[1]The fact that actions are not deterministic will not be addressed in detail in this paper.

Agents' decision-making strategies are based upon personal, rather than group-wide, objectives. If an agent is willing to cooperate, she may be unable to construct a plan to do so; an agent who is unwilling or unable to assist can propose an alternative that the original requester may now contemplate adopting. After agreeing to cooperate, an agent can "opt out" at any time, without obligation to notify other agents.

When cooperation is first established during communication, the agents must determine how they will coordinate. Sometimes, nothing needs to be done to begin coordinating; more often, though, the requester will idle for several time steps — for example, if a lifter is not currently ready to lift the box. An agent will stop waiting if another agent initiates communication, either to establish a new agreement or to indicate progress on a current agreement (e.g., the other lifter now indicates she is ready to act). The agent will also stop waiting upon observing the completion of the request (e.g., the box appears on the hand-truck). Finally, if an agent is idle "too" long (i.e., longer than a pre-set threshold), the agent inquires about the status of her request (possibly discovering that the other agent has opted out).

Communication is the only mechanism whereby agents can check if they are working on the same goals since there are no global structures, such as blackboards, available. While observation is sufficient to engineer the exit from coordination problems, the agents are not assumed to possess the common goals and knowledge of each other required in order to solve coordination problems without any communication [cf. Genesereth *et al.*, 1986; Huber and Durfee, 1995].

3 Leveraging past experience

This section will describe the case-based reasoning [Kolodner, 1993] techniques that individual agents use to acquire and use coordinated procedures in order to better coordinate. A guiding principle of these techniques is that memory should be organized around coordination points. There is no communication between the agents during the learning process; the memories are created and maintained by each agent independently.

3.1 Learning coordinated procedures

Coordination points influence three facets of the learning process. Most importantly, coordination devices that represent past successful joint achievement of coordination points form the skeleton of future plans. Next, memories are primarily indexed based on expected future requests in order to make the memory more likely to be recalled during communication. Third, coordination points influence the determination of state-based secondary indices.

Figure 3 contains pseudo-code for LEARNCOORDINATEDPROCEDURE, which is the method by which agents transform experience into memories. Coordinated procedures are derived from execution traces, which are quite noisy because actions and requests can fail or be ineffective for other reasons. There are many details involved in the process of transforming such noisy data into something suitable for learning that have not been addressed in prior work on procedural learning. Space prevents covering many of them here; the interested

```
LEARNCOORDINATEDPROCEDURES (executionTrace) ≡
  cleanTrace ← CLEAN(executionTrace)
  segmentGoalsPairs ← SEGMENT(cleanTrace)
  forall ⟨segment, goals⟩ in segmentGoalsPairs
    coreSegment ← REMOVEINEFFICIENCIES(segment, goals)
    STORECOORDINATEDPROCEDURE(coreSegment, goals)

STORECOORDINATEDPROCEDURE (coreSegment, goals) ≡
  procedure ← LIST()
  times ← LIST()
  requests ← LIST()
  forall step in REVERSE(coreSegment)
    if KEEP(step, procedure)
      procedure ← PUSH(step, procedure)
      times ← LIST()
      requests ← LIST()
    times ← PUSH(STARTTIME(step), times)
    if step is an agreement to achieve a coordination point
      requests ← PUSH(step, requests)
  if HEAD(procedure) is not a member of requests
    ADDCASEBASEENTRY(procedure, goals, null, times)
  forall r in requests
    rTimes ← REMOVELATERTHAN(times, STARTTIME(r))
    ADDCASEBASEENTRY(procedure, goals, r, rTimes)
```

Figure 3: Algorithms to learn coordinated procedures.

reader is directed elsewhere [Garland, 2000] for more about cleaning, segmenting, and removing inefficiencies from execution traces. There are two other non-trivial procedures, KEEP and ADDCASEBASEENTRY, whose pseudo-code is not given but whose important aspects will be described below.

STORECOORDINATEDPROCEDURE converts an execution trace segment into a coupling of a coordinated procedure and its indexing information. The primary tasks of STORECOORDINATEDPROCEDURE are:

1. Construct a coordinated procedure by summarizing and optimizing the execution trace segment.
2. Determine the set of expected requests that the agent could use as retrieval cues in the future.
3. For each entry to be added to the case base, determine the set of states in which the agent should consider retrieving the entry.

The motives for summarizing an execution trace segment are: the particular time at which subordinate goals were achieved at runtime may be misleading; the stored plan will be more easily adapted in the future (at the cost of regenerating the original action if it is needed again); and the state-based indices for case-base entries are generalized each time the same procedure is stored under different indices. Optimizations allow agents to improve upon, rather than just repeat, the way in which some coordination points are jointly achieved.

Storing procedures under several indices is a good heuristic when agents do not share indexing information and there is uncertainty about the setting at the outset of cooperation. In addition, the state of the world at the start of the first step of

the coordinated procedure is not the only reasonable benchmark for determining future settings in which the procedure will be effective. Alternative indexing states are provided by the (removed) steps in the execution trace summary that precede the first kept step.

The pseudo-code in Figure 3 sets forth how STORECOORDINATEDPROCEDURE records indexing information while adding actions to the coordinated procedure. A call to the KEEP function determines which actions from the execution trace segment are added to the procedure. For each step, whether it is kept or not, the time at which it was started is added to a list of indexing times and agreements are added to a list of expected requests. When a step is added to the procedure, these lists are reset.

After the coordinated procedure has been determined, it is stored into memory without any reference to an expected request (unless the first step in the plan is an expected request). Next, the procedure is stored again for each expected request so that the request is part of the primary storage index. Indexing in this way facilitates retrieving coordinated procedures during conversations. The means by which coordination points influence the state-based secondary indices is less direct, and is determined by ADDCASEBASEENTRY.

ADDCASEBASEENTRY makes representational changes to the procedure and maintains the structure of the underlying case base. In terms of the organization of memory, a notable division is that STORECOORDINATEDPROCEDURE computes the relevant times to check the state of the world; ADDCASEBASEENTRY converts those states into concrete case-base indices. For this work, the secondary indexing scheme is influenced by the top-level goals and the expected requests of the entry. The literals from these are culled and then all relevant predicates in the state relating to the literals are identified. Relevancy is determined from the surface features of the environment, rather than from analysis of the stored plan [cf. Hammond, 1990; Veloso and Carbonell, 1993].

The most interesting aspect of KEEP is the treatment of coordination points. The principle summarization criteria is to remove actions that are planner-reconstructible — this produces a skeleton of coordination mechanisms that is fleshed out by the essential individual actions needed to support the achievement of coordination. An important characteristic of summarization is that it never removes: (1) coordination mechanisms for agreements, (2) coordination points that correspond to unexpected requests, or (3) actions that end a shift between goals. Requests for joint action require special handling to ensure that the initiator only keeps the joint action and the assistant only remembers to expect a request.

The heuristic optimizations currently implemented are intended to eliminate some conversations altogether. For example, based on past experience, a lifter learns to load the hand-truck without being explicitly told to do so. Likewise, the hand-truck operator learns to expect the lifter to load the hand-truck without the hand-truck operator's guidance. The risk of the heuristics is that if agents do not recall compatible memories, time and effort may be wasted.

One optimization rule (implemented in KEEP) is that an agent who agreed to a request removes the corresponding coordination mechanism from the coordinated procedure. This guideline reflects an optimistic belief that the agent knows the right time to accomplish the request without being specifically asked. A second guideline, dual to the first, is part of the representational changes performed by ADDCASEBASEENTRY. For an initiator of a request, the procedure passed into ADDCASEBASEENTRY contains a coordination mechanism that will prompt the agent to establish the same request again in the future. This mechanism is heuristically converted into an (optimistic) expectation that the request for service will be satisfied without a direct request. These two optimizations currently only relate to requests for service and not to requests for joint action, which require more precise timing.

```
Time         Action                          Outcome

  1  agreed with L2 to ...                  Summarized
 21  <LIFT-TOGETHER XLBOX1>                 Summarized
 91  HTO asked to (ON XLBOX1 HANDTR3)       Optimized
105  L2 agreed to ...                       Summarized
126  <LOAD-TOGETHER XLBOX1 HANDTR3>         Kept
161  <LIFT MBOX0>                           Summarized
191  <CARRY MBOX0 STREET1>                  Summarized
231  <LOAD MBOX0 TRUCK3>                    Kept
266  HTO asked to (ON XLBOX1 TRUCK3)        Optimized
279  L2 agreed to ...                       Summarized
300  <UNLOAD-TOGETHER XLBOX1 HANDTR3>       Summarized
335  L2 agreed to ...                       Summarized
353  <LOAD-TOGETHER XLBOX1 TRUCK3>          Kept

L1 adding case-base entry MEM75
Procedure: (<LOAD-TOGETHER ?L1-177 ?L1-174>
            <LOAD ?L1-175 ?L1-176>
            <LOAD-TOGETHER ?L1-177 ?L1-176>)
Top-level goals:  ((ON ?L1-177 ?L1-176)
                   (ON ?L1-175 ?L1-176))
Request: NIL
State indices based on ticks 1, 21, 91, 105, 126

L1 adding case-base entry MEM76 derived from MEM75
Request: (LIFT-TOGETHER ?L1-177) by ?L1-180
State indices based on tick 1

L1 adding case-base entry MEM77 derived from MEM75
Request: (ON ?L1-177 ?L1-174) by ?L1-173
State indices based on ticks 1, 21, 91
```

Figure 4: Three entries for a single coordinated procedure.

Figure 4 shows some (lightly edited) output when agent L1 is adding multiple case-base entries for the same coordinated procedure. In this example, the agent is creating one top-level entry and two request entries. The performance that led to learning this procedure was exceptional (33% fewer ticks than average) and was chosen for illustrative purposes; execution traces are normally much more chaotic. The only failure, which is not shown in the figure, is an attempt at time 71 by the two lifters to jointly carry the box to the street.

3.2 Acting from shared past experience

Acting from shared past experience can lead agents to coordinate more efficiently. For example, when an agent receives a familiar request, she can retrieve a plan which anticipates future requests rather than merely creating a plan to satisfy that single request. When different agents anticipate the same

points of coordination, they can coordinate more effectively for three reasons:

1. They will not waste time discussing alternatives that will prove to be unproductive.
2. They will not waste time negotiating over two viable alternatives.
3. In some cases, they can eliminate communication entirely.

Unfortunately, acting from past experience does not guarantee that agents will coordinate more efficiently. First of all, there may not be regularity in the types of coordination problems faced by the community. Second, agents will sometimes assess the same situation in disparate manners. This reflects both differences in experience between agents and the open-endedness of interpretation in general. Thus, it is important to store procedures so that different agents are likely to retrieve compatible memories in many situations. In this paper, compatible viewpoints on which past activities to recall develop when storage is guided by coordination points and surface features of the environment.

When planning, agents prefer coordinated procedures from similar past activities to plans constructed by the baseline planner. However, an agent does not search her case base when a planning session immediately follows the failure of a retrieved plan. For the experiments in the next section, coordinated procedures are recalled during planning slightly more than 20% of the time during the last problem-solving episode.

An agent measures the similarity of a case-base entry to the current setting by determining what percentage of the stored predicates can be made true given the possible mappings of literals in the current state to required role-fillers for the entry. For those entries that meet or exceed the current highest similarity (at least above a pre-set threshold of 0.50), the coordinated procedure stored in the entry is checked to see if it can be adapted to the current state of the world. If so, the current highest similarity is updated. Finally, the best of the best is selected from the entries with maximal similarity. There are various ways to determine the best of the best such as randomly or by the number of storages for the entry. The results in the next section rank the plans using the same heuristic as the baseline planner (most time-efficient) and break any remaining ties randomly.

In experiments conducted so far, the number of entries in each agent's case base has been less than 50, so neither storage nor retrieval was a bottleneck. In general, a more refined secondary indexing scheme, such as indexing by differences [Kolodner, 1983a,b] might be needed. Further refinements such as distinguishing between short and long term memory or selectively "forgetting" past cases [Smyth and Keane, 1995] may also be fruitful.

4 Empirical results and analysis

This section supports the claim that learning coordinated procedures leads agents to better coordinate with empirical studies from an implemented testbed. The experimental methodology takes into account the possible influence of sampling bias and ordering effects; each data point reported is the average of 600 trials.

In all of the experiments in this section, agents acquire coordination knowledge independently of learning coordinated procedures — agents learn more accurate probability estimates of the likelihood of success of possible actions. Agents use probabilities when planning from scratch, when deciding who to ask for help, when deciding whether to cooperate, and when adapting coordinated procedures to match the current problem setting. Having more accurate probability estimates, therefore, can cause agents to behave more efficiently. A learning structure, based on COBWEB [Fisher, 1987] trees, enables agents to increase the accuracy of probability estimates by generalizing past experiences interacting with the domain and other agents. For more details, see Garland [2000].

There are many ways to measure the performance of the agent community, such as the number of primitive actions attempted and the number conversations that occur. The best overall measure of community effort, however, is the number of ticks of a simulated clock that transpire during the course of the community solving the problem. The number of ticks measures both action and communication effort, in addition to time when the agents are idle for one reason or another (see Figure 2).

In order to measure the advantages of learning coordinated procedures under various conditions, the type of coordinated procedures learned and the initial ability of agents to solve coordination problems were varied. The possible types of coordinated procedures are:

Basic A basic coordinated procedure is not optimized and only achieves a single goal.

Improved An improved coordinated procedure is optimized and may achieve multiple goals (as in Figure 4).

A third possibility is to learn better probability estimates only and not learn any coordinated procedures.

The three initial levels of ability to solve coordination problems are:

Minimal The minimal amount of coordination knowledge is a predicate-based communication language based on goals, coordination points, and the ability to make unambiguous external references.

Basic Basic agents have additional planning knowledge of other agents and the ability to make goal-selection choices based on past experience. The coordination information built into basic agents was determined by analyzing the coordination knowledge implicitly acquired by basic coordinated procedures.

Expert Expert agents have additional hand-crafted goal-selection and coordination strategies, including an extension of the heuristic optimizations.

Note that basic and expert agents have abilities that exceed those assumed in Section 2.

Table 1 presents a tabular summary of the results of running the system for each of the combinations of initial ability and type of coordinated procedures learned. For each of these nine runs, the chart shows the number of ticks required to solve the tenth problem-solving episode faced by the community. The data in the first column indicate that improving

Built-in Coordination Ability	Type of Coordinated Procedures Learned		
	None	Basic	Improved
Minimal	846.3	747.2	623.5
Basic	789.2	743.5	640.6
Expert	687.4	641.5	594.1

Table 1: Comparison of the amount of community effort required for the tenth problem-solving episode.

the initial coordination ability of non-learning agents substantially reduces the number of ticks. Scanning across each row of Table 1 shows that, regardless of the initial coordination ability of the agents, learning basic coordinated procedures led to significant reductions in the number of ticks (with 99% confidence). Improved coordinated procedures are always significantly more effective than basic ones.

The second row of Table 1 warrants special attention. By design, agents with basic initial coordination abilities have access to the same goal-selection and planning knowledge as agents can learn over time through basic coordinated procedures. The fact that agents with basic abilities nonetheless benefit from learning such procedures shows that these pieces of knowledge are more useful as a unit, when retrieved from the case base, than when accessed separately.

The benefits of learning coordinated procedures are immediate. This is evinced by the learning curves, plotted with their 99% confidence intervals, shown in Figure 5. A comparison of curves MN and EN, two runs when agents are not learning coordinated procedures, points out that the impact of augmenting the initial ability to solve coordination problems does not diminish over time. A comparison of curves MI and MN, two runs when agents have minimal initial abilities to solve coordination problems, shows that the importance of learning coordinated procedures grows over time. Most importantly, comparing curves MI and EN reveals that learning coordinated procedures is more effective than initial expertise by the second problem-solving episode.

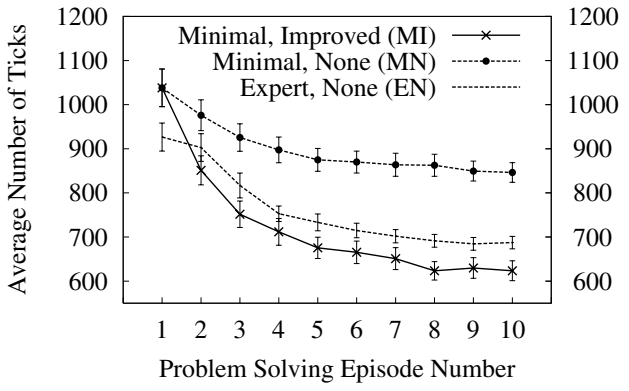


Figure 5: Comparing runtime effort.

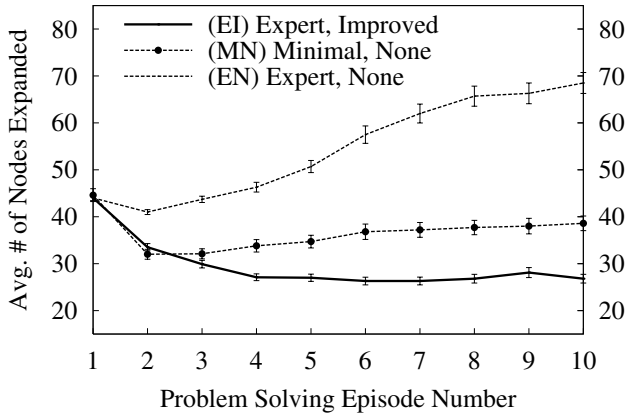


Figure 6: Comparing planning effort.

Another significant advantage of learning coordinated procedures is in controlling the amount of planner search. Figure 6 depicts planning effort, as measured by the average number of planning nodes expanded per call to the planner. The disparity between curves MN and EN provides evidence that initial expertise comes with a price — the increased amount of planning information can lead to a large increase in planner search. However, learning and acting from past experience controls planner search so effectively that experts who learn coordinated procedures (curve EI) expanded fewer planning nodes than agents with minimal initial knowledge who do not learn them (curve MN).[2]

Learning coordinated procedures leads to similar statistical improvement in the number of conversations, the number of ticks spent conversing, the number of attempted actions, and the number of successful actions. Furthermore, these results hold across a wide spectrum of possible goal-selection strategies, cooperation strategies, and communication costs [Garland, 2000].

Overall, the results clearly demonstrate that coordinated procedures are an effective resource for agents to learn to better coordinate their run-time activities. Learning coordinated procedures benefits agents regardless of the initial ability to solve coordination problems. There is leverage in storing and retrieving plans based on the surface features of the environment, rather than having access to similar information that is accessed at separate times. Finally, the techniques are very effective in preventing increased planner search.

5 Related Research

Learning coordination knowledge in multi-agent systems has been studied in frameworks with different assumptions than those made in this paper, such as in homogeneous systems [Sugawara and Lesser, 1998] and communication-free domains [Haynes and Sen, 1998; Ho and Kamel, 1998]. In a system with similar underpinnings, Prasad and Lesser [1999] implement a learning system that extends the generalized partial global planning [Decker and Lesser, 1992] architecture

[2]CPU time, another common measure of planning effort, is also reduced significantly by learning coordinated procedures.

by allowing the agents to choose a coordination mechanism (from a commonly known set of choices) based upon the results of training runs. All agents make the same choice because agents communicate their local viewpoints to all other agents to form a consistent global viewpoint and each agents records the same training data. By contrast, in this paper, each agent learns independently on the basis of their own experiences. Also, the techniques are applicable in the absence of built-in common knowledge and agents acquire procedures in addition to compatible viewpoints about how to coordinate activity.

In this work, procedural learning is based on run-time behavior, which differs from learning techniques based upon the output of planning sessions [Carbonell, 1983; Veloso and Carbonell, 1993; Laird *et al.*, 1986; Kambhampati and Hendler, 1992; Sugawara, 1995]. Reusing a plan derivation will not produce a sequence of actions to better solve a similar problem in the future if the derivation was deficient (due to the agent's incomplete knowledge). On the other hand, execution traces encapsulate the history of both planned and unplanned agent interactions with the domain. Consequently, procedures are learned that were not developed in a single (or multiple) planning histories. Thus, some coordinated procedures stored in memory can represent unplanned successes. Remembering examples of past successes differs from previous approaches to changing run-time behavior that have emphasized learning from failures [Hammond, 1990; Haynes and Sen, 1998].

References

Richard Alterman and Andrew Garland. Convention in joint activity. *Cognitive Science*, 25(4), 2001.

Jaime Carbonell. Derivational analogy and its role in problem solving. In *Proc. Third National Conference on Artificial Intelligence*, pages 64–69, 1983.

Keith S. Decker and Victor R. Lesser. Generalized partial global planning. *Intl. Journal of Intelligent and Cooperative Information Systems*, 1(2):319–346, 1992.

Marie E. desJardins, Edmund H. Durfee, Charles L. Ortiz, and Michael J. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20(4):13–22, 1999.

Edmund H. Durfee and Victor R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, 1991.

Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

Andrew Garland. *Learning to Better Coordinate in Joint Activities*. PhD thesis, Brandeis University, 2000.

Michael Genesereth, Matt Ginsberg, and Jeffrey Rosenschien. Cooperation without communication. In *Proc. Fifth National Conference on Artificial Intelligence*, pages 51–57, 1986.

Barbara Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–357, 1996.

Barbara Grosz and Candace Sidner. Plans for discourse. In Philip R. Cohen, Jerry Morgan, and Martha E. Pollack, editors, *Intentions in Communication*, pages 417–444. Bradford Books, 1990.

Kristian J. Hammond. Case-based planning: A framework for planning from experience. *Cognitive Science*, 14:385–443, 1990.

Thomas Haynes and Sandip Sen. Learning cases to resolve conflicts and improve group behavior. *Intl. Journal of Human-Computer Studies*, 48:31–49, 1998.

Fenton Ho and Mohamed Kamel. Learning coordination strategies for cooperative multiagent systems. *Machine Learning*, 33(2-3):155–177, 1998.

Marcus J. Huber and Edmund H. Durfee. Deciding when to commit to action during observation-based coordination. In *Proc. First Intl. Conference on Multiagent Systems*, pages 163–170, 1995.

Subbarao Kambhampati and James A. Hendler. Control of refitting during plan reuse. *Artificial Intelligence*, 55:193–258, 1992.

Janet L. Kolodner. Maintaining organization in a dynamic long-term memory. *Cognitive Science*, 7:243–280, 1983.

Janet L. Kolodner. Reconstructive memory: A computer model. *Cognitive Science*, 7:281–328, 1983.

Janet L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

John E. Laird, Paul S. Rosenbloom, and Alan Newell. Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

Hector J. Levesque, Philip R. Cohen, and José H. T. Nunes. On acting together. In *Proc. Eighth National Conference on Artificial Intelligence*, pages 94–99, July 1990.

M. V. Nagendra Prasad and Victor R. Lesser. Learning situation-specific coordination in cooperative multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 2:173–207, 1999.

Barry Smyth and Mark T. Keane. Remembering to forget. In *Proc. Fourteenth Intl. Joint Conference on Artificial Intelligence*, pages 377–382, 1995.

Toshiharu Sugawara and Victor Lesser. Learning to improve coordinated actions in cooperative distributed problem-solving environments. *Machine Learning*, 33(2-3):129–153, 1998.

Toshiharu Sugawara. Reusing past plans in distributed planning. In *Proc. First Intl. Conference on Multiagent Systems*, pages 360–367, 1995.

Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

Manuela Veloso and Jaime Carbonell. Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, 10:249–278, 1993.

MULTI-AGENT SYSTEMS

MULTI-AGENT SYSTEMS

Fair Imposition

Yoav Shoham and Moshe Tennenholtz *
Computer Science Department
Stanford University
Stanford, CA 94305

Abstract

We introduce a new notion, related to auctions and mechanism design, called *fair imposition*.[1] In this setting a center wishes to fairly and efficiently allocate tasks among a set of agents, not knowing their cost structure. As in the study of auctions, the main obstacle to overcome is the self-interest of the agents, which will in general cause them to hide their true costs. Unlike the auction setting, however, here the center has the power to impose arbitrary behavior on the agents, and furthermore wishes to distribute the cost as fairly as possible among them. We define the problem precisely, present solution criteria for these problems (the central of which is called *k-efficiency*), and present both positive results (in the form of concrete protocols) and negative results (in the form of impossibility theorems) concerning these criteria.

1 Introduction

Allocation of resources or, isomorphically, of tasks is among the fundamental problems in computer science, operations research, economics, and other scientific and technological disciplines. In a centralized task allocation problem there is a center whose aim is to allocate one or more tasks among several available agents (be they machines, processors, servers, employees, companies, etc.). Several aspects of the situation strongly impact the problem. The two aspects we focus on are:

1. The information available to the center about the agents' costs for performing the tasks.
2. The center's ability to impose tasks and payments on the agents

In classical work on centralized optimization in CS and OR the assumption is that the center has perfect information about and perfect control over the agents; usually payments don't figure in at all, and the center imposes on the agents an optimal protocol computed by the center. In economics and game theory for the most part the opposite obtains, namely the center (auctioneer, procurer) has no knowledge of the agents' private information, and furthermore has no power to enforce any behavior on the agents. Indeed, the individual freedom to decide whether to transact and under what terms to do so lies at the core of what one usually understands a 'market' to mean. For this reason, payments are the primary means of inducing agents to exhibit any sort of behavior in such a setting.

We introduce an intermediate setting in which the center has full power over the agents, but no access to their private information. We furthermore assume that the center is a benign dictator, which wishes not only to achieve the tasks but to do so in a way that is socially fair. The problem is that agents will in general not volunteer correct information that would allow the center to achieve these objectives, and thus the center must resort to incentive engineering of the sort encountered in mechanism design in game theory.

This broad category of problems was motivated by a specific problem encountered by the military, in the Virtual Transportation Company [VTC] project [Godfrey and Mifflin, 2000]. In most if not all democratic countries the state has the power at times of emergency to commandeer aircraft and other resources required to deal with the crisis. Of course, in a democratic country the state recognizes the rights of companies such as airline carriers, and attempts to compensate them for such use. The problem is how to decide which carriers to tap, and how to compensate the parties affected. Ideally, the state would like to achieve the following:

1. Acquire the required types and number of aircraft.
2. Minimize (in some cases, eliminate) its own costs.
3. Minimize the total true costs to the carriers tapped.
4. Distribute the cost fairly among all the carriers, those tapped and not.

*The permanent address of the second author is: Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel

[1]This work was supported by DARPA grant F30602-00-2-0598-P00001

While our treatment of the problem going forward will be entirely abstract, and indeed make some assumptions that are not consistent with this application (in particular, that the military wishes to pay less than market value for the services rendered), the reader might keep this example in mind throughout the paper.

The rest of the paper is organized as follows. In section 2 we define the procurement problem. The procurement problem is isomorphic to the problem of task allocation (or the auctioning of a good). In section 3 we discuss the fair imposition of tasks. In particular, we introduce the notion of k-efficiency, which is central for the evaluation of procurement protocols in the context of fair allocation. Sections 4–8 present a set of basic results dealing with the feasibility of fair allocation of tasks. We present several upper bounds (by means of impossibility results) on the level of fairness and the minimization of expenses that can be obtained, as well as protocols for fair task allocation. Further discussion of the contribution of our work in the context of general task allocation and protocols for non-cooperative environments can be found in section 9.

2 The procurement problem

In this section we provide the reader with the requisite knowledge of the task allocation/procurement theory. Basic procurement theory has much similarity with basic auction theory. Following the related basic procurement/auction theory literature, we restrict ourselves to a 1-shot, single-item procurements problem. We later (see Section 8) weaken the second restriction, and in future work deal with the first one.

Consider a center who wishes to obtain a particular service, where there are n potential suppliers, or agents, denoted by $1, 2, \ldots, n$ who may supply this service. A procurement protocol is a procedure in which participants submit messages (typically monetary bids) for providing the service. The protocol's rules specify the type of messages, and as a function of the messages submitted by the participants they determine the service provider and the payments to be made by the participants. The payments may be positive, negative, or zero.

Formally, a procurement procedure for n potential participants, $N = \{1, 2, \ldots, n\}$ is characterized by 4 parameters, M, g, c, d, where M is the set of messages, $g = (g_1, \ldots, g_n)$ with $g_i : M^n \rightarrow [0, 1]$ for all i and $\sum_{i=1}^{n} g_i(m) \leq 1$ for all m, and $c = (c_1, \ldots, c_n); d = (d_1, \ldots, d_n)$ with $c_i, d_j : M^n \rightarrow R$ for all i, j. The interpretation of these elements is as follows. Participant i submits a message $m_i \in M$. Let $m = (m_1, m_2, \ldots, m_n)$ be a vector of messages, then the organizer conducts a lottery to determine the service provider, in which the probability that i is the winner equals $g_i(m)$. The winner, say j, is paid $c_j(m)$ and every other participant i pays $d_i(m)$. The classical theory of procurement associates a (Bayesian) game with each procurement procedure and analyses the behavior of the agents under the equilibrium assumption, as described below. Each agent has a type, v_i, selected from a set of possible types V. The type of agent i, v_i, is known to it, but might not be known to the other agents. The type v_i should be interpreted as the cost for agent i in providing the required service. A strategy for agent i is a function $b_i : R_+ \rightarrow M$, where $b_i(v_i)$ is the message submitted by i when his type is v_i. Each agent i has a utility function u_i. Assuming that the agents submitted the tuple of messages $m = (m_1, m_2, \ldots, m_n)$, the service will be allocated based on g. The utility of agent i will be $c_i(m) - v_i$ if it has been selected as the service provider, and otherwise its utility will be $-d_i(m)$. A tuple of strategies, $b = (b_1, b_2, \ldots, b_n)$ will be called an equilibrium, if for every agent i, b_i is the best response against the strategies of the other agents in b (denoted by b_{-i}). A strategy b_i of agent i will be called a dominant strategy if for any tuple of strategies b_{-i} of the other agents, and for any strategy b'_i of agent i we have that $u_i(b_i, b_{-i}) \geq u_i(b'_i, b_{-i})$.

3 From procurement to fair imposition

In the setup discussed in this paper, the center can force the agents to provide a desired service, as well as monetary payments. However, the center may wish to get the desired service while attempting to minimize the agents' costs.

Definition 1 *Given a procurement setting, with n possible service providers, $N = \{1, 2, \ldots, n\}$, and costs selected from the set V, a procurement protocol (M, g, c, d), where $M = V$ is the set of messages (possibly declared costs), g is the allocation function, c determines the service provider payments, and d determines the other agents' payments, is called* incentive compatible *if for any cost $v_i \in V$ of agent i, its payoff is maximized by sending the message $m_i = v_i$ regardless of the messages of the other agents (i.e., truth revealing is a dominant strategy).*

In the sequel we will restrict ourselves to incentive compatible protocols. Incentive compatible protocols have several desired properties. In particular, they do not build on any rationality assumption on the behavior of other agents. This implies that an agent can refrain from adopting and computing probabilistic information about the other agents' behavior when employing a dominant strategy.

In order to introduce a basic definition of fair imposition of protocols we use the following idea. We wish first to ensure that the center will minimize its expenses due to the performance of the service. Given that, we would like to minimize the expenses of each and every agent. This leads to the following central definition:

Definition 2 *Given a procurement setting S with n agents, a procurement protocol P is called* k-efficient *if the following holds:*

1. *P is incentive compatible.*
2. *For any tuple of costs $(v_1, v_2, \ldots, v_n)$ where v_i is the cost of agent i, we have that:*

(a) *The sum of payments from the center to the agents is non-positive.*

(b) *The cost to agent i is no more than $\frac{v_{[k]}}{n}$, where $v_{[k]}$ is the k order statistics of $\{v_1, v_2, \ldots, v_n\}$.*[2]

Notice that our definition of fairness captures the desire to minimize expenses of each particular agent. Notice that we have required that the center's budget be non-negative. The case where the procurer expects to pay for its services is discussed in Section 7. In the sequel we will denote the agent with the i-th lowest valuation by a_i and its valuation by $v_{[i]}$.

4 1-efficiency

Work on economic mechanism design in game theory [Fudenberg and Tirole, 1991] usually adopts several basic requirements. One of these basic requirements is that protocols should be economically efficient. We will use the following standard definition:

Definition 3 *Given a procurement setting S with n agents, a procurement protocol P will be called economically efficient if it always selects the agent with the lowest cost to serve as the service provider.*

We can now prove:

Lemma 1 *Given a procurement setting S with n agents, a procurement protocol P is 1-efficient only if it is economically efficient.*

Proof (sketch): Assume that the protocol chooses the agent with the second lowest valuation as the service provider. Recall we denote the agent with the i-th lowest valuation by a_i and its valuation by $v_{[i]}$. In order to get 1-efficiency agents $a_1, a_3, a_4, \ldots, a_n$ need to suffer a cost of at most $\frac{1}{n}v_{[1]}$. This implies that if a_2 is the service provider, then it will suffer an expense of at least $v_{[2]} - \frac{n-1}{n}v_{[1]} > \frac{1}{n}v_{[1]}$, which contradicts 1-efficiency. Similar argument holds when the agent who will provide the service is $a_j, j \geq 2$.

The above lemma teaches us that in order to have 1-efficiency we must have economic efficiency. However, we now show:

Lemma 2 *There is no protocol that is both 1-efficient and economically efficient.*

Proof (sketch): In order to have 1-efficiency agents $a_2, a_3, \ldots, a_n$ should suffer an expense of at most $\frac{v_{[1]}}{n}$. Since the center may not have a negative balance, and since we require 1-efficiency, and since the cost of the service for a_1 is $v_{[1]}$ we get that the payments are as follows: each agent $a_i, i \geq 2$ pays exactly $\frac{1}{n}v_{[1]}$ to the center, who collects these payments and transfers that to a_1. However, such incentive compatible efficient protocol, where the sum of payments is 0, does not exist (see [Mas-Colell *et al.*, 1995], page 880).[3]

[2] The k order statistic of a set is the k lowest element in this set.

[3] This is no longer the case if we consider Bayesian equilibrium [d'Aspremont and Gerard-Varet, 1979].

Hence, as we have seen, economic efficiency is essential for 1-efficiency, but 1-efficiency and economic efficiency are contradicting in our setup. Hence, we get:

Theorem 1 *There is no 1-efficient procurement protocol.*

5 2-efficiency

Given the impossibility of 1-efficiency the next desirable alternative is to consider the case of 2-efficiency. As before, we are first interested in the connections between 2-efficiency and economic efficiency. We can show:

Lemma 3 *Given a procurement setting S with n agents, a procurement protocol P is 2-efficient only if it is economically efficient.*

Proof (sketch): Assume that the protocol assigns the good to agent a_2. Notice that in order to obtain 2-efficiency all other agents will have to pay exactly $\frac{1}{n}v_{[2]}$. This implies that agent 2 will suffer an expense of $\frac{1}{n}v_{[2]}$. Hence, a_2 may go ahead and report a cost v', where $v_{[1]} < v' < v_{[2]}$. Consider now the tuple of types $(v_{[1]}, v', v_{[3]}, \ldots, v_{[n]})$. Regardless of who is the agent to be allocated the service for this valuation (whether this is agent a_2 or another agent), 2-efficiency will imply that agent a_2 will suffer an expense of no more than of $\frac{1}{n}v'$. Hence, the above deviation is rational, which contradicts incentive compatibility. It is immediate to see that an allocation of the service to agent $a_j, j \geq 3$ can not lead to 2-efficiency. Hence, the allocation needs to be efficient.

Thus again we'd like to know whether economic efficiency and 2-efficiency are compatible. Unfortunately, at this time we don't, so instead we give a weaker result. Consider the following property defined on procurement protocols.

Definition 4 *Given a procurement setting S with n agents, a procurement protocol is called* unbiased, *if for every tuple of agents' types the payments by all agents who do not provide the service are identical, and the following property* does not *hold:*

- *For any tuple of agents' types, the service provider's expenses are greater than or equal to the expenses of all other agents, and for at least one tuple of types its expense is greater than the expenses of all other agents.*

Thus a biased procurement protocol either favors the agents that do not provide the service, or differentiates among them. We can now show:

Theorem 2 *Given a procurement setting S with n agents, there is no unbiased 2-efficient procurement protocol.*

Proof (sketch): First, given the previous lemma we can restrict our attention to protocols that guarantee efficient allocation. Notice that for any tuple of types, either all agents' expenses are exactly $\frac{1}{n}v_{[1]}$, or there should be at least one agent who suffers expenses of more than $\frac{1}{n}v_{[1]}$ (otherwise the center will suffer losses). Since

the protocol is unbiased there must be at least one tuple of types, for which agent a_2 pays $\frac{1}{n}v_{[1]}+\epsilon$, for some $\epsilon > 0$. Assume that for this tuple of types, a_2 reports the cost $v_{[1]}+\delta$, where $\delta > 0$ and $\frac{v_{[1]}+\delta}{n} < \frac{1}{n}v_{[1]}+\epsilon$ (which is satisfied for any $\delta < n\epsilon$). We get that, since the allocation needs to be economically efficient (and therefore this modification will not change the allocation of service) and since the protocol is 2-efficient, such deviation will decrease the expense of agent a_2: consider the behavior of the protocol on the tuple $(v_{[1]}, v_{[1]}+\delta, v_{[3]}, \ldots, v_{[n]})$, we get that agent a_2 will pay no more than $\frac{v_{[1]}+\delta}{n} < \frac{1}{n}v_{[1]}+\epsilon$. This contradicts incentive compatibility. This implies that there is no unbiased 2-efficient protocol.

6 3-efficiency

The negative results of the previous sections teach us that in order to obtain fair imposition of services we need to consider protocols that are at most 3-efficient. In this section we show that this upper bound can be matched by an appropriate protocol. Consider the following protocol.

Fair3:

1. Each supplier is asked to reveal its costs to the center.
2. The task will be allocated to the agent who has announced the lowest cost; this agent will be paid the second lowest cost announced.
3. Each supplier will pay to the center $\frac{1}{n}$ of the second lowest reported cost of the other participants.

Notice that this protocol (as well as protocol Fair3b to be discussed in section 8) makes use of the Groves scheme for mechanism design [Groves, 1973]. We can now show:

Proposition 1 *Given a procurement setting S with n agents, Fair3 is a 3-efficient protocol for that setting.*

Proof (sketch) : Notice that if every agent reports its actual costs, then the payments by a_1 and a_2 are $\frac{v_{[3]}}{n}$, and the payment by $a_j, j \geq 3$ is $\frac{v_{[2]}}{n}$. We need to show that truth revealing is a dominant strategy. However, since the payments here fit the Groves scheme (see [Groves, 1973]) we get that the protocol is also incentive compatible.

As we can see, if we are willing to settle for 3-efficiency, then quite fair task allocation for minimizing the agents' actual expenses can be obtained. The center will be able to obtain the desired behavior without suffering any cost.

7 Almost budget balanced protocols

Given the general results of the previous sections, it may be of interest to see how far we are from obtaining 2-efficiency. In order to address this issue we challenge one of the assumptions of our model, namely that in no case is any cost imposed on the center. Indeed, while in some situations the center should be expected to pay nothing, in many others they expect to bear some costs. In some cases - including, arguably, the VTC domain, which motivated this work –the expectation is that the service providers experience a positive surplus. Here we however explore the intermediate situation in which the center is willing to suffer some expense, but a minimal one.

In order to handle the above issue let us assume that $V = [a, b]$ where $b > a$, i.e. the agents' costs are in between a and b. In many domains, assuming the costs are high, we have that $b - a << a$. For example, the costs to various airlines of providing a given flight might range from \$750K to \$800K. The center be willing to suffer a payment of \$800K-\$750K=\$50K but not of \$750K or more.

Given the above intuition we consider the following definition:

Definition 5 *Given a procurement setting with n agents, a procurement protocol P will be called an almost budget balanced k-efficient protocol, if the following holds:*

1. *P is incentive compatible.*
2. *For any tuple of costs $(v_1, v_2, \ldots, v_n)$ where v_i is the cost of agent i, we have that:*
   (a) *The sum of payments from the center to the agents is at most $v_{[n]} - v_{[1]}$.*
   (b) *The payment by agent i is no more than $\frac{v_{[k]}}{n}$, where $v_{[k]}$ is the k order statistics of $\{v_1, v_2, \ldots, v_n\}$.*

Consider the following protocol:

almost2:

1. Each supplier is asked to reveal its cost to the center.
2. The task will be allocated to the agent who has announced the lowest cost; this agent will be paid the second lowest cost announced.
3. Each supplier will pay to the center $\frac{1}{n}$ of the lowest reported cost of the other participants.

We can now show:

Proposition 2 almost2 *is an almost budget balanced 2-efficient protocol for any given procurement setting.*

Proof (sketch): Let there be n agents in the setting with valuations $v_{[1]}, ..., v_{[n]}$. Since the payment scheme fits the Groves payments (see [Groves, 1973]) we get that the protocol is incentive compatible. In addition, if every agent reports its actual valuation then the payment by a_1 is $\frac{v_{[2]}}{n}$, and the payment by $a_j, j \geq 2$ is $\frac{v_{[1]}}{n}$. The center will pay $\frac{n-1}{n}v_{[2]} - \frac{n-1}{n}v_{[1]} < v_{[n]} - v_{[1]}$.

8 Extension: the imposition of interacting services

Our study in the previous sections has concentrated on the imposition of a single task on a set of agents. If there are several tasks, we can deal with each of those tasks separately, and apply the techniques and results

we previously obtained. Although this may be quite appropriate in many cases, one may wish to consider more general extensions. In this section, we briefly consider one extension. We will concentrate on the case of *two interacting service.*

Consider two services, 1 and 2. The cost of performing service j by agent i will be denoted by $v_i(j)$, and the cost of performing both services by agent i will be denoted by $v_i(\{1,2\})$. The fact that there exists some interaction between the services will be captured by the fact that it might be that $v_i(\{1,2\}) \neq v_i(1) + v_i(2)$. For example, a carrier's cost for a pair of flights might be lower than the sum of costs for each of the flights since these flights might refer to two consecutive periods or two consecutive routes (notice the similarity with combinatorial auctions, a popular topic in the recent AI literature [Fujishima *et al.*, 1999; Sandholm, 1999; Tennenholtz, 2000]; in both cases the valuation for a pair of goods might be different from the sum of valuations of these goods).

The procurement setting definition will be now revised to have two services, and n agents with costs/types as above, selected from a set V. For ease of exposition let $V = [0, b]$ for some $b > 0$.

Definition 6 *Given a procurement setting with two services, and with n possible service providers, $N = \{1, 2, \ldots, n\}$, where costs for single services and for the pair of services are selected from the set V, a procurement protocol (M, g, c, d) is a tuple where $M = V^3$ is the set of messages, and a message $m = (m_1, m_2, m_3)$ declares the costs for service 1,2, and the pair $\{1,2\}$ respectively, where g is the allocation function, c determines the payments to the service providers, and d determines the other agents' payments. Such a protocol is called incentive compatible if for every valuation $v_i(1), v_i(2), v_i(\{1,2\}) \in V$ of agent i, its payoff is maximized by sending the message $m_i = (v_i(1), v_i(2), v_i(\{1,2\}))$ regardless of the messages of the other agents.*

The definition of k-efficiency can be extended in various ways in order to handle the case of two interacting services. We now describe one of these possible extensions.

Definition 7 *Given a procurement setting S with two interacting services, and n agents, a procurement protocol P will be called k-efficient, if the following holds:*

1. *P is incentive compatible.*
2. *For any tuple of costs of the agents we have:*
   (a) *The sum of payments from the center to the agents is non-positive.*
   (b) *The payment by agent i is no more than $\frac{v_{[k]}(1)+v_{[k]}(2)}{n}$, where $v_{[k]}(j)$ is the k order statistics of $\{v_1(j), v_2(j), \ldots, v_n(j)\}$*

It is easy to extend our infeasibility results for 1-efficiency and for 2-efficiency to the case of two interacting services. As we will now show, the positive result on 3-efficiency can be generalized as well.

Consider the following protocol:

Fair3b:

1. Each supplier is asked to reveal its costs to the center.
2. The tasks will be allocated to the agents who have announced the lowest cost; notice that both tasks can be allocated to the same agent, or the tasks can be allocated to different agents.
3. If a supplier s has been selected to supply both services then he will be paid as follows: an allocation of the lowest cost possible, ignoring this supplier's messages will be calculated, and the supplier s will be paid according to the cost associated with this allocation.
4. If a supplier s has been selected to supply the service x, and another supplier s' has been selected to supply the service y, then s will be paid as in (3), minus the cost associated with supplying y by s'.
5. For each agent i we consider the second lowest declared cost, $c_i(j)$ of the other agents for service j. Agent i will be asked to pay to the center $\frac{c_i(1)+c_i(2)}{n}$.

We can now show:

Proposition 3 *Given a procurement setting S with two interacting services and n agents, Fair3b is a 3-efficient protocol for that setting.*

9 Discussion

Social systems face the challenge of distributing efforts among service providers, in a way that will obtain the society goals, while attempting to minimize costs for the individuals in that society. Therefore, the problem of fair imposition of tasks appears in a variety of domains, and is fundamental to obtaining efficient and fair procurement procedures. In this paper we have introduced a general rigorous setting, where the fair imposition of tasks can be studied. Using this setting, we have provided general results on the fair imposition of services in multi-agent systems.

Our study deals with the problem of task allocation with self-motivated service providers, where the center can enforce agents' actions (e.g. their payments). The complexity of this setting stems from the fact that the participants can try and cheat the center about their private information (e.g. their costs) while the center wishes to keep the allocation *fair* and to minimize the expenses of each individual participant. As a result, our study complements previous work on social laws (e.g. [Moses and Tennenholtz, 1995; Shoham and Tennenholtz, 1995]) and on the imposition of protocols on multi-agent systems (e.g. [Minsky, 1991a; 1991b]), as well as work in information economics (see [Kreps, 1990] for a general discussion). Our work also complements work in Distributed AI dealing with rules on interactions for self-motivated agents (e.g. [Rosenschein and Zlotkin, 1994; Kraus, 1997]), as well as

work bridging the gap between Game Theory and Computer Science [Boutilier *et al.*, 1997; Tennenholtz, 1999; Varian, 1995]. Perhaps the most relevant line of research is work on optimal auction design (which is isomorphic to work on optimal design of procurement protocols; see [Wolfstetter, 1996; McAfee and McMillan, 1987; Milgrom, 1987] for overviews). However, our setting, where fairness is the major objective (rather than economic efficiency or center's revenue) and behaviors can be enforced (hence, participation constraints do not apply), distinguishes our line of research from the economic mechanism design literature.

We see the procurement setting as a basic building block for general task allocation in multi-agent systems. Hence, in order to address general task allocation in non-cooperative environments we need to obtain deep understanding of the basic procurement setup. This seems essential for the design of protocols for non-cooperative environments. Needless to say, when facing this fundamental need, basic work in AI dealing with protocols for non-cooperative environments and work on mechanism design in game theory share much in common. The notion of fair imposition and its study are the contributions of this paper to these lines of research.

Much left to be done. In particular, the study of the multi-item (i.e. interacting services) setting should be further developed. In addition, an explicit treatment of repeated procurement situations is a challenge of considerable importance. More specific challenge is to try and come with a biased 2-efficient protocol. We plan to pursue these extensions in our future work. We believe that the study of the fair imposition of tasks is a challenge of fundamental importance to the design of effective multi-agent systems, and hope that others will join us in this effort.

References

[Boutilier *et al.*, 1997] C. Boutilier, Y. Shoham, and M.P. Wellman. Special issue on economic principles of multi-agent systems. *Artificial Intelligence*, 94, 1997.

[d'Aspremont and Gerard-Varet, 1979] C. d'Aspremont and L.A. Gerard-Varet. Incentives and incomplete information. *Journal of Public Economics*, 11(1):25–45, 1979.

[Fudenberg and Tirole, 1991] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

[Fujishima *et al.*, 1999] Y. Fujishima, K. Leytontractable-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJCAI-99*, 1999.

[Godfrey and Mifflin, 2000] G.A. Godfrey and T.L. Mifflin. Virtual transportation company challenge problem. DARPA working paper, http://www.task-program.org, 2000.

[Groves, 1973] T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

[Kraus, 1997] S. Kraus. Negotiation and cooperation in multi-agent environments. *Artificial Intelligence*, 94, 1997.

[Kreps, 1990] D. Kreps. *A Course in Microeconomic Theory*. Princeton University Press, 1990.

[Mas-Colell *et al.*, 1995] A. Mas-Colell, M.D. Whinston, and J.R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[McAfee and McMillan, 1987] R.P. McAfee and J. McMillan. Auctions and bidding. *Journal of Economic Literature*, 25:699–738, 1987.

[Milgrom, 1987] P.R. Milgrom. Auction theory. In T. Bewly, editor, *Advances in Economic Theory: Fifth World Congress*. Cambridge University Press, 1987.

[Minsky, 1991a] N.H. Minsky. The imposition of protocols over open distributed systems. *IEEE Transactions on Software Engineering*, 17(2):183–195, 1991.

[Minsky, 1991b] N.H. Minsky. Law-governed systems. *Software Engineering Journal*, pages 285–302, September 1991.

[Moses and Tennenholtz, 1995] Y. Moses and M. Tennenholtz. Artificial Social Systems. *Computers and Artificial Intelligence*, 14(6):533–562, 1995.

[Rosenschein and Zlotkin, 1994] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter*. MIT Press, 1994.

[Sandholm, 1999] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *IJCAI-99*, 1999.

[Shoham and Tennenholtz, 1995] Y. Shoham and M. Tennenholtz. Social Laws for Artificial Agent Societies: Off-line Design. *Artificial Intelligence*, 73, 1995.

[Tennenholtz, 1999] M. Tennenholtz. Electronic commerce: From game-theoretic and economic models to working protocols. In *IJCAI-99*, 1999.

[Tennenholtz, 2000] M. Tennenholtz. Some tractable combinatorial auctions. Proceedings of AAAI-2000, 2000.

[Varian, 1995] H.R. Varian. Economic mechanism design for computerized agents. Working paper, Berkeley University, 1995.

[Wolfstetter, 1996] E. Wolfstetter. Auctions: An introduction. *Journal of Economic Surveys*, 10(4):367–420, 1996.

Robust Multi-unit Auction Protocol against False-name Bids

Makoto Yokoo, Yuko Sakurai, and **Shigeo Matsubara**
NTT Communication Science Laboratories
2-4 Hikaridai, Seika-cho
Soraku-gun, Kyoto 619-0237 Japan
email: {yokoo, yuko, matsubara}@cslab.kecl.ntt.co.jp
url: http://www.kecl.ntt.co.jp/csl/ccrg/members/{yokoo, yuko, matubara}

Abstract

This paper presents a new multi-unit auction protocol (IR protocol) that is robust against false-name bids. Internet auctions have become an integral part of Electronic Commerce and a promising field for applying agent and Artificial Intelligence technologies. Although the Internet provides an excellent infrastructure for executing auctions, the possibility of a new type of cheating called false-name bids has been pointed out. A false-name bid is a bid submitted under a fictitious name.

A protocol called LDS has been developed for combinatorial auctions of multiple different items and has proven to be robust against false-name bids. Although we can modify the LDS protocol to handle multi-unit auctions, in which multiple units of an identical item are auctioned, the protocol is complicated and requires the auctioneer to carefully predetermine the combination of bundles to obtain a high social surplus or revenue. For the auctioneer, our newly developed IR protocol is easier to use than the LDS, since the combination of bundles is automatically determined in a flexible manner according to the declared evaluation values of agents. The evaluation results show that the IR protocol can obtain a better social surplus than that obtained by the LDS protocol.

1 Introduction

Internet auctions have become an especially popular part of Electronic Commerce (EC). The Internet provides an excellent infrastructure for executing much cheaper auctions with many more sellers and buyers from all over the world. However, in [Sakurai *et al.*, 1999], the authors pointed out the possibility of a new type of cheating called *false-name bids*, i.e., an agent may try to profit from submitting false bids made under fictitious names, e.g., multiple e-mail addresses. Such a dishonest action is very difficult to detect since identifying each participant on the Internet is virtually impossible. Compared with collusion [Rasmusen, 1994; Varian, 1995], a false-name bid is easier to execute since it can be done by someone acting alone, while a bidder has to seek out and persuade other bidders to join in collusion.

Auctions can be classified into three types by the number of items/units auctioned: (i) single item, single unit, (ii) single item, multiple units, and (iii) multiple items. In [Sakurai *et al.*, 1999; Yokoo *et al.*, 2000a], we analyzed the effects of false-name bids on auction protocols. The obtained results can be summarized as follows.

- For multi-unit auctions, where the demand of a participant can be multiple units, or for combinatorial auctions of multiple items, the generalized Vickrey auction protocol (GVA) [Varian, 1995] is not robust against false-name bids.
- There exists no auction protocol that simultaneously satisfies incentive compatibility, Pareto efficiency, and individual rationality for all cases in the above situations if agents can submit false-name bids.

In this paper, we concentrate on private value auctions [Mas-Colell *et al.*, 1995]. In private value auctions, each agent knows its own evaluation values of goods, which are independent of the other agents' evaluation values. We define an agent's utility as the difference between the true evaluation value of the allocated goods and the payment for the allocated goods. Such a utility is called a *quasi-linear* utility [Mas-Colell *et al.*, 1995]. These assumptions are commonly used for making theoretical analyses tractable.

In a traditional definition [Mas-Colell *et al.*, 1995], an auction protocol is (dominant strategy) incentive compatible, if bidding the true private values of goods is the dominant strategy for each agent, i.e., the optimal strategy regardless of the actions of other agents. The revelation principle states that in the design of an auction protocol we can restrict our attention to incentive compatible protocols without loss of generality [Mas-Colell *et al.*, 1995; Yokoo *et al.*, 2000a]. In other words, if a certain property (e.g., Pareto efficiency) can be achieved using some auction protocol in a dominant strategy equilibrium, i.e., the combination of dominant strategies of agents, the property can also be achieved using an incentive compatible auction protocol.

In this paper, we extend the traditional definition of incentive-compatibility so that it can address false-name bid manipulations, i.e., we define that an auction protocol is (dominant strategy) incentive compatible, if bidding the true private values of goods by using the true identifier is the dominant strategy for each agent. Also, we say that auction pro-

tocols are robust against false-name bids if each agent cannot obtain additional profit by submitting false-name bids. If such robustness is not satisfied, the auction protocol lacks incentive compatibility.

We say an auction protocol is Pareto efficient when the sum of all participants' utilities (including that of the auctioneer), i.e., the social surplus, is maximized in a dominant strategy equilibrium. In a more general setting, Pareto efficiency does not necessarily mean maximizing the social surplus. In an auction setting, however, agents can transfer money among themselves, and the utility of each agent is quasi-linear; thus the sum of the utilities is always maximized in a Pareto efficient allocation.

An auction protocol is individually rational if no participant suffers any loss in a dominant strategy equilibrium, i.e., the payment never exceeds the evaluation value of the obtained goods. In a private value auction, individual rationality is indispensable; no agent wants to participate in an auction where it might be charged more money than it is willing to pay.

In this paper, we concentrate on multi-unit auctions, in which multiple units of an identical item are sold. Multi-unit auctions have practical importance and are widely executed already in current Internet auction sites such as eBay, Yahoo!. In current Internet auctions, a participant is assumed to want only one unit of an item. By allowing a participant to bid on multiple units, e.g., he/she needs two units of the item at the same time, as in combinatorial auctions [Sandholm, 1999; Fujishima *et al.*, 1999; Lehmann *et al.*, 1999], we can increase both the utility of the participants and the revenue of the seller.

The GVA protocol [Varian, 1995] is one instance of the well-known Clarke mechanism [Mas-Colell *et al.*, 1995]. It satisfies incentive compatibility, Pareto efficiency, and individual rationality in multi-unit auctions when there exists no false-name bid; however, this protocol is not robust against false-name bids [Sakurai *et al.*, 1999].

If the marginal utility of a unit always decreases for all agents, the GVA is robust against false-name bids [Sakurai *et al.*, 1999]. The marginal utility of an item means an increase in the agent's utility as a result of obtaining one additional unit. If the number of units becomes very large, the marginal utility of a unit tends to decrease. For example, if we already have one million units of an item, the utility of having additional one unit would be close to zero. On the other hand, if the number of units are relatively small, which is common in many auction settings, we cannot assume that the marginal utility of each agent always decreases. A typical example where the marginal utility increases is an all-or-nothing case, where an agent needs a certain number of units, otherwise the good is useless (e.g., airplane tickets for a family trip).

In [Yokoo *et al.*, 2000b], we developed a combinatorial auction protocol called the Leveled Division Set (LDS) protocol. This protocol satisfies incentive compatibility and individual rationality even if agents can submit false-name bids. We can modify the LDS protocol so that it can handle multi-unit auctions. As far as the authors know, this is the only existing non-trivial multi-unit auction protocol that is robust against false-name bids. However, this protocol is complicated and requires the auctioneer to carefully pre-determine the combination of bundles in order to obtain a high social surplus or revenue.

In this paper, we develop a new multi-unit auction protocol that satisfies incentive compatibility and individual rationality. We call this protocol the Iterative Reducing (IR) protocol. In this protocol, the combination of bundles is automatically determined in a flexible manner according to the declared evaluation values of agents.

In the following, we first show how the LDS protocol can be modified to handle multi-unit auctions. Then, we describe the details of the IR protocol and prove that it satisfies incentive compatibility. Furthermore, we compare the obtained social surplus of the IR protocol with that of the LDS protocol.

2 Leveled Division Set Protocol

We show how the LDS protocol can be modified for multi-unit auctions. In the following, we define several terms and notations. To help readability, we use two different types of parentheses to represent sets: $\{\}$ and [].

- A set of agents: $\{1, 2, \ldots, N\}$
- The number of units of an item: M
- A declared evaluation value of agent i for j units of an item: $b_{i,j}$ (which may or may not be true).
- An auctioneer determines a reservation (minimal) price r for one unit of an item.
- A reservation price of a bundle of j units is $r \times j$.
- A division D is defined as a set of bundles $\{m_1, m_2, \ldots\}$, where $\sum_l m_l \leq M$. For example, a division $\{8, 2\}$ means that the auctioneer is going to sell a bundle of 8 units and a bundle of 2 units.

Also, an auctioneer determines a *leveled division set*. Figure 1 shows examples of leveled division sets. Case 1 shows one instance where there exist three units of an item, and cases 2 and 3 show instances where there exist ten units of an item. A leveled division set is defined as follows.

- Levels are defined as $1, 2, \ldots, max\_level$.
- For each level k, a division set $SD_k = [D_{k1}, D_{k2}, \ldots]$ is defined so that the sum of multiple bundles that appear in a division must appear in an earlier level. For example, In case 2 of Figure 1, there is a division $\{1,1,1,1,1,1,1,1,1,1\}$ in level 3. Therefore, the sum of the multiple bundles of this division, i.e., $2, 3, \ldots$, or 10, appears in earlier levels.

To execute the LDS protocol, the auctioneer must predefine the leveled division set and the reservation price of one unit r. Each agent i declares $b_{i,j}$ for all $1 \leq j \leq M$. The winners and payments are determined by calling the procedure LDS(1). LDS(k) is a recursive procedure defined as follows.

Procedure LDS(k)

Step 1: If there exists only one agent i whose evaluation value of a bundle of j units satisfies $b_{i,j} \geq r \times j$,

	case 1	case 2	case 3
level 1	[{3}]	[{10}]	[{10}]
level 2	[{2,1}]	[{9}, {8,2}, {7,3},{6,4},{5,5}]	[{9}, {8}, {7}, {6,4},{5,5}]
level 3		[{1,1,1,1,1,1,1,1,1,1}]	[{3,3,3}]
level 4			[{2,2,2,2,2}]
level 5			[{1,1,1,1,1,1,1,1,1,1}]

Figure 1: Example of Leveled Division Sets

and j is included in an element of SD_k, then compare the results obtained by the procedure GVA(k) and by LDS$(k+1)$, and choose the one that gives the larger utility for agent i.

Step 2: If there exist at least two agents i, i' whose evaluation values $b_{i,j}, b_{i',j'}$ of bundles of j and j' units satisfy $b_{i,j} \geq r \times j$ and $b_{i',j'} \geq r \times j'$, and j, j' are included in elements of SD_k, then apply the procedure GVA(k).

Step 3: Otherwise: call LDS$(k+1)$, or terminate if $k = max\_level$.

Procedure GVA(k): For a division $D = \{m_1, m_2, \ldots\}$ and one possible allocation of units G, we say G is allowed under D if each bundle in D is allocated to different agents in G. Also, we allow that some bundles are not allocated to any agent. In that case, we assume that the bundles are allocated to the auctioneer (denoted by 0), whose evaluation value for a bundle of j units is equal to $r \times j$. The declared evaluation value of agent i for an allocation G (represented as $e_i(G)$) is defined as $b_{i,j}$, if j units are allocated to agent i in G, otherwise $e_i(G) = 0$. Choose an allocation G^* such that it is allowed under the divisions in SD_k and it maximizes $\sum_{x \in \{0,1,\ldots,N\}} e_x(G)$. The payment of agent i (represented as p_i) is calculated as $\sum_{x \neq i} e_x(G^*_{\sim i}) - \sum_{x \neq i} e_x(G^*)$, where $G^*_{\sim i}$ is the allocation that is allowed under the divisions in SD_k and maximizes the sum of all agents' (including the auctioneer 0) evaluation values except that of agent i. This procedure is basically identical to the GVA.

Example 1 *Let us assume there exist ten units of an item. The evaluation values of agents are defined as follows.*

	1	2	3	> 3
agent 1	0	0	33	33
agent 2	0	0	32	32
agent 3	0	0	31	31
agent 4	14	14	14	14
agent 5	13	13	13	13
agent 6	12	12	12	12
agent 7	11	11	11	11
...				
agent 13	11	11	11	11

In this case, the evaluation values of agent 1,2, and 3 are all-or-nothing, i.e., they need three units at once. Other agents need only one unit, and agents $7, 8, \ldots, 13$ *have identical evaluation values. Also, let us assume the reservation price for one unit is 10 and the leveled division set is defined as case 3 in Figure 1. Since there exists no evaluation value that is larger than the reservation price in levels 1 and 2, units are sold using the level 3 division. As a result, each of the agents 1,2, and 3 obtains a bundle of three units, and each pays the reservation price 30.*

The intuitive explanation that the LDS protocol is robust against false-name bids is as follows. Let us assume that an agent uses two identifiers, and obtains one unit by each identifier in the level 5 of case 3. If this agent uses a single identifier, it can obtain a bundle of two units in the level 4 by paying the reservation price, which is the minimum price for obtaining two units.

3 IR Protocol

3.1 Limitations of LDS Protocol

One limitation of the LDS protocol is that the auctioneer must pre-define the possible combinations of bundles as a leveled division set. To increase the social surplus or the revenue, the auctioneer must choose an appropriate leveled division set, but this task is not easy. Even if the auctioneer has some knowledge of the distributions of agents' evaluation values, the auctioneer must solve a very complicated optimization problem to find an appropriate leveled division set.

Also, in the LDS protocol, if there exists an evaluation value that is larger than the reservation price for any single bundle in the level, the auctioneer must sell units using the current level. For example, in the situation of Example 1, when the reservation price for a unit is 10 and the leveled division set of case 2 in Figure 1 is used, there exists an evaluation value for a bundle of three units. Therefore, the auctioneer must sell units using the divisions of level 2. In this case, only three units can be sold, although there exist many participants whose evaluation values for a single unit are larger than the reservation price.

When there are many demands for smaller bundles, such as for two or three units, using the leveled division set of case 2 in Figure 1 is inappropriate. If we use the leveled division set of case 3, the result would be better, but we still must pre-define the possible combinations of bundles and cannot flexibly combine bundles according to demands.

3.2 Overview of IR Protocol

In our newly developed Iterative Reducing (IR) protocol, instead of determining the possible divisions of units in advance, we determine the allocations of bundles sequentially

from larger bundles. More specifically, as in the LDS protocol, we first check whether there exists an agent whose evaluation value for a bundle of M units is larger than the reservation price $r \times M$. If not, we reduce the number of units in a bundle one by one. When some bundles of k units are allocated, and there exist enough remaining units, we allocate smaller bundles.

3.3 Details of IR Protocol

The protocol is executed by calling the procedure IR$(M, M, \{1, 2, \ldots, N\})$, which is defined as follows.

Procedure IR $(m, j,$ Participants$)$

Step 1: When $j = 0$, terminate the protocol.

Step 2: Set k to the largest integer where $j \times k \leq m$ holds. Set Candidates $\leftarrow \{i \mid i \in$ Participants$, b_{i,j} \geq r \times j\}$, $n \leftarrow |$Candidates$|$.

Step 3: When $n > k$:

- **3.1:** Set Winners to a set of agents i, where $i \in$ Candidates and $b_{i,j}$ is within k-th highest evaluation values in Candidates, and set p to the $k+1$-th highest evaluation value in Candidates (ties are broken randomly).
- **3.2:** Each agent $i \in$ Winners gets a bundle of j units, and pays p. Terminate the protocol.

Step 4: When $n = k$:

- **4.1:** Set Winners $\leftarrow$ Candidates, and $p \leftarrow r \times j$.
- **4.2:** For each $i \in$ Winners, compare its utility for obtaining a bundle of j units by paying p, and that for the result of IR$(m - j \times (n-1), j-1,$ (Participants$-$ Winners$) \cup \{i\})$, and choose the one that gives the higher utility for agent i. Terminate the protocol.

Step 5: When $n < k$:

- **5.1:** Set Winners $\leftarrow$ Candidates, and $p \leftarrow r \times j$.
- **5.2:** For each $i \in$ Winners, compare its utility for obtaining a bundle of j units by paying p, and that for the result of IR$(m - j \times (n-1), j-1,$ (Participants$-$ Winners$) \cup \{i\})$, and choose the one that gives the higher utility for agent i.
- **5.3** Call IR$(m - j \times n, j - 1,$ Participants $-$ Winners$)$.

When the result of IR$(m - j \times (n-1), j - 1,$ (Participants $-$ Winners$) \cup \{i\})$ is used in Step 4.2 or Step 5.2, although we calculate the allocated units and payment of agent i as if agents other than Winners could obtain some units, we don't assign any units nor transfer money to agents except Winners.

3.4 Examples of Protocol Application

Example 2 *Let us assume there are 12 units of an item, and the reservation price of one unit is 10. The evaluation values of agents are identical to Example 1. From IR*$(12, 12, \{1, 2, \ldots, 13\})$ *to IR*$(12, 4, \{1, 2, \ldots, 13\})$*, n becomes* 0 *in Step 2. In IR*$(12, 3, \{1, 2, \ldots, 13\})$*, the condition of Step 5 holds. Each of the agents 1,2, and 3 obtains a bundle of three units and pays the reservation price 30. Then, IR*$(3, 2, \{4, \ldots, 13\})$ *is called. In IR*$(3, 1, \{4, \ldots, 13\})$*, the condition of Step 3 holds, and each of the agents 4, 5, and 6 obtains one unit bundle and pays 11.*

Example 3 *Let us assume there are 12 units of an item and the reservation price of one unit is 10. The evaluation values of agents are defined as follows.*

	< 4	4	5	> 5
agent 1	0	0	52	52
agent 2	0	51	51	51
agent 3	0	48	48	48

In IR$(12, 5, \{1, 2, 3\})$*, the condition of Step 4 holds. Each of the agents 1 and 2 obtains a bundle of five units at the reservation price 50. However, in the procedure of Step 4.2, agent 2 prefers the result of IR*$(7, 4, \{2, 3\})$*, i.e., obtaining a bundle of four units at 48. Thus, this result is applied for agent 2.*

Example 4 *The procedures in Steps 4.2 and 5.2, which compare the results and choose the one that gives higher utility for an agent, are necessary to guarantee incentive compatibility.*

Let us assume we omit these procedures. Then, in the situation of Example 3, agent 2 obtains a bundle of five units when it truthfully declares its evaluation values and its utility is $51 - 50 = 1$*. When agent 2 understates its evaluation values as 49 for a bundle of four and five units, then agent 2 can obtain a bundle of four units and its utility becomes* $51 - 48 = 3$*. Thus, for agent* 2*, declaring its true evaluation value cannot be a dominant strategy without these procedures.*

Example 5 *In Steps 3 and 4, even if there exist remaining units, the protocol is terminated. This might seem wasteful, but it is necessary to guarantee incentive compatibility. Let us assume the protocol is continued as long as there exists at least one remaining unit. In the situation of Example 3, agent 3 cannot obtain any unit when it truthfully declares its utility, thus its utility is 0. On the other hand, agent 3 can use a false name agent 4 and changes its declarations as follows.*

	1	2	3	4	5	> 5
agent 1	0	0	0	0	52	52
agent 2	0	0	0	51	51	51
agent 3	0	24	24	24	53	53
agent 4	0	24	24	24	53	53

Then, in IR$(12, 5, \{1, 2, 3, 4\})$*, the condition of Step 3 holds, and each of the agents 3 and 4 obtains a bundle of five units. When the protocol is continued as long as there exists at least one remaining unit, we must apply a similar comparison procedure in Step 3 as well as in Steps 4 and 5. Since these agents prefer obtaining a bundle of two units, each agent obtains a bundle of two units by paying the reservation price 20. In reality, agent 3 obtains four units by paying 40 and its utility is* $48 - 40 = 8$*, thus using a false-name bid is profitable.*

In the IR protocol, when a bundle of j units are sold, and the number of remaining units is smaller than j, the protocol will be terminated. Therefore, an agent cannot obtain a fraction of units at a lower price nor gather a fraction of units by using multiple identifiers.

4 Proof of Incentive Compatibility

We are now going to prove the following theorem.

Theorem 1 *The IR protocol satisfies incentive compatibility.*

To prove this theorem, we use the following lemmas.

Lemma 1 *If an agent i uses two identifiers i' and i'', and obtains a bundle of x and y units under the identifiers i' and i'', respectively, then agent i can obtain $z = x + y$ units at a price that is less than (or equal to) the sum of the prices for x and y by using a single identifier.*

The proof is as follows. For the price of x units p_x and the price of y units p_y, $p_x \geq r \times x$ and $p_y \geq r \times y$ hold. Since agent i obtains x and y units, $M \geq z$ holds.

Now, let us consider the situation where agent i (or i', i'') does not participate in the auction. When selling bundles of z units, let us assume the condition of Step 3 or Step 4 holds. In this case, when another agent participates, the agent cannot obtain a bundle smaller than z. This is because the protocol is terminated before or just after selling bundles of z units. This contradicts the assumption that agent i' obtains a bundle of x units and i'' obtains a bundle of y units. Thus, when bundles of z units are sold, the condition of Step 5 must hold.

Now, let us assume that agent i participates and declares its evaluation value for z units $b_{i,z}$ as $r \times z + \epsilon$, where ϵ is a very small amount. Since without i's participation, the condition of Step 5 holds. By adding this declaration, the condition of Step 4 or Step 5 holds. In either case, agent i can obtain a bundle of z units with a payment of $p = r \times z$, which is smaller than (or equal to) $p_x + p_y$, i.e., the sum of the payments when agent i uses multiple identifiers. □

Using a similar method of the proof of Lemma 1, we can show that if an agent uses more than two identifiers, it can obtain the same number of units with a smaller (or equal) payment by using a single identifier.

So far, we have shown that an agent cannot increase its utility by using false-name bids. Now, we are going to show that truth-telling is the dominant strategy for each agent under the assumption that the agent uses a single identifier.

Lemma 2 *An agent cannot increase its utility by overstating its evaluation value.*

The proof is as follows. Assume that agent i's true evaluation value of j units is $v_{i,j}$, and it overstates its evaluation value as $b_{i,j}$, where $b_{i,j} > v_{i,j}$. It is clear that if agent i cannot be in Winners by declaring $b_{i,j}$ when selling bundles of j units, overstating is useless. Furthermore, if agent i is in Winners when it declares $v_{i,j}$, since its payment is independent from its declared evaluation value, overstating is useless.

The only possible case where overstating might be effective is when agent i declares $v_{i,j}$, it is not in Winners, and when it declares $b_{i,j}$, it becomes a part of Winners. However, in this case, the payment becomes larger than the true evaluation value $v_{i,j}$. Therefore, to obtain a positive utility, the procedures of Step 4.2 or Step 5.2 must be applied and agent i obtains a bundle smaller than j. However, the situations considered in these procedures are identical to the situation when agent i truthfully declares its utility as $v_{i,j}$. Thus, overstating is useless. □

Lemma 3 *An agent cannot increase its utility by understating its evaluation value.*

The proof is as follows. Assume that agent i's true evaluation value of j units is $v_{i,j}$, and it understates its evaluation value as $b_{i,j}$, where $b_{i,j} < v_{i,j}$. It is clear that if agent i cannot be in Winners when declaring $v_{i,j}$, understating is useless. Furthermore, if agent i is still in Winners when it declares $b_{i,j}$, since its payment is independent from its declared evaluation value, understating is useless.

The only possible case where understating might be effective is when agent i declares $v_{i,j}$, it is in Winners, and when it declares $b_{i,j}$, it can be excluded from Winners. However, in this case, if the condition of Step 3 holds when agent i truthfully declares $v_{i,j}$, by understating, either the condition of Step 3 or Step 4 holds. In both cases, the protocol is terminated and agent i cannot obtain any units. On the other hand, let us assume the condition of Step 4 or 5 holds when agent i truthfully declares $v_{i,j}$. Then, the situation that occurs when agent i declares $b_{i,j}$ is identical to the situation considered in Step 4.2 or Step 5.2 when agent i truthfully declares $v_{i,j}$. If agent i prefers this result, the result is applied when agent i truthfully declares $v_{i,j}$. Thus, understating is useless. □

From these lemmas, we can derive Theorem 1.

5 Evaluations

In this section, we compare the obtained social surplus of the IR protocol and that of the LDS protocol using a simulation. We determine the evaluation values of agent i by the following method. This method is based on the binomial distribution used in [Fujishima *et al.*, 1999] for evaluating winner determination algorithms in combinatorial auctions.

- Determine the size of a bundle j that agent i wants to have by using a binomial distribution $B(M, p)$, i.e., the probability that the size of the bundle is j is given by $p^j(1-p)^{M-j}M!/(j!(M-j)!)$.
- Randomly choose $v_{i,j}$, i.e., i's evaluation value for the bundle of j, from within the range of $[0, j]$. We assume that the evaluation values of an agent are all-or-nothing, i.e., the evaluation value for a bundle smaller than j is 0. Also, we set the evaluation value for a bundle larger than j to $v_{i,j}$, i.e., having additional units is useless.

We generated 100 problem instances by setting the number of agents $N = 10$, the number of units $M = 10$, and $p = 0.2$. Figure 2 shows the average ratio of the obtained social surplus to the Pareto efficient social surplus by varying the reservation price. We show the results of the IR protocol and those of the LDS protocol, in which the leveled division sets are case 2 and case 3 in Figure 1. Since we set $M = 10$ and $p = 0.2$, the size of a bundle is most likely to be 2, and 1 and 3 are also likely to exist. Thus, the leveled division set of case 3 seems to be a reasonable choice.

If the GVA is used and agents cannot submit false-name bids, then the obtained social surplus is Pareto efficient and the ratio becomes 100%. On the other hand, when agents can submit false-name bids, we cannot predict the obtained result of the GVA, since declaring true evaluation values is no longer a dominant strategy for each agent.

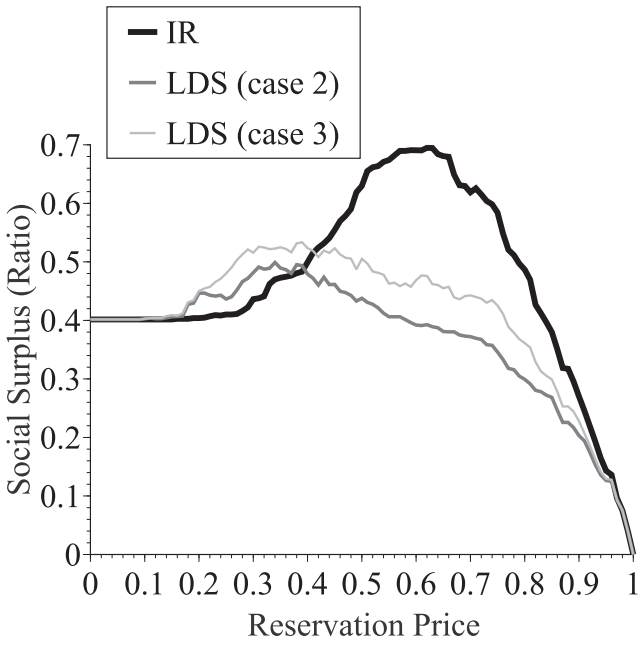


Figure 2: Comparison of Social Surplus

As shown in Figure 2, by setting the reservation price to an appropriate value, the social surplus of the IR protocol becomes about 70% of the Pareto efficient social surplus. On the other hand, in the LDS protocol, the social surplus becomes at most 53% of the Pareto efficient social surplus. In both protocols, when the reservation price is small, one bundle of M units is sold to a single agent. By increasing the reservation price, the units are divided among multiple agents. In the IR protocol, the combination of bundles is determined in a flexible manner, and by increasing the reservation price, the protocol can select the combination of bundles that increases the obtained social surplus. On the other hand, in the LDS protocol, the possible combinations of bundles must be pre-defined; thus the protocol cannot adjust to various problem instances. We have obtained very similar results for the problem instances generated using different parameter settings and different distributions of agents' evaluation values.

If the auctioneer has some knowledge of the distributions of agents' evaluation values, optimizing the reservation price for the IR protocol would be relatively easy compared with optimizing both the reservation price and the leveled division set for the LDS protocol. Of course, we cannot say that the IR protocol always achieves a better social surplus than that of the LDS. If agents require very large bundles, then the IR protocol might end up selling only $m + 1$ units out of $M = 2m$ units. In such a case, the LDS protocol with a carefully designed leveled division set could beat the IR protocol.

6 Conclusions

In this paper, we developed a multi-unit auction protocol (IR protocol) that is robust against false-name bids. Compared with the LDS protocol, this protocol is easier for an auctioneer to use, since he/she only needs to determine the reservation price of one unit, while he/she must pre-define possible combinations of bundles as well as the reservation price in the LDS protocol. We showed that the IR protocol can obtain a much better social surplus than that obtained by the LDS protocol, since it can determine the combination of bundles in a flexible manner according to the declared evaluation values of agents. Our future works will include developing combinatorial and double auction protocols [Yokoo *et al.*, 2001] that are robust against false-name bids based on the IR protocol.

References

[Fujishima *et al.*, 1999] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computation complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 548–553, 1999.

[Lehmann *et al.*, 1999] Daniel Lehmann, Liadan Ita O'Callaghan, and Yoav Shoham. Truth revelation in approximately efficient combinatorial auction. In *Proceedings of the First ACM Conference on Electronic Commerce (EC-99)*, pages 96–102, 1999.

[Mas-Colell *et al.*, 1995] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[Rasmusen, 1994] Eric Rasmusen. *Games and Information*. Blackwell, 1994.

[Sakurai *et al.*, 1999] Yuko Sakurai, Makoto Yokoo, and Shigeo Matsubara. A limitation of the Generalized Vickrey Auction in Electronic Commerce: Robustness against false-name bids. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*, pages 86–92, 1999.

[Sandholm, 1999] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auction. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 542–547, 1999.

[Varian, 1995] Hal R. Varian. Economic mechanism design for computerized agents. In *Proceedings of the First Usenix Workshop on Electronic Commerce*, 1995.

[Yokoo *et al.*, 2000a] Makoto Yokoo, Yuko Sakurai, and Shigeo Matsubara. The effect of false-name declarations in mechanism design: Towards collective decision making on the Internet. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS-2000)*, pages 146–153, 2000.

[Yokoo *et al.*, 2000b] Makoto Yokoo, Yuko Sakurai, and Shigeo Matsubara. Robust combinatorial auction protocol against false-name bids. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, pages 110–115, 2000.

[Yokoo *et al.*, 2001] Makoto Yokoo, Yuko Sakurai, and Shigeo Matsubara. Robust double auction protocol against false-name bids. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-2001)*, 2001. (to appear).

Bundle Design in Robust Combinatorial Auction Protocol against False-name Bids

Makoto Yokoo, Yuko Sakurai, and **Shigeo Matsubara**
NTT Communication Science Laboratories
2-4 Hikaridai, Seika-cho
Soraku-gun, Kyoto 619-0237 Japan
email: {yokoo, yuko, matsubara}@cslab.kecl.ntt.co.jp
url: http://www.kecl.ntt.co.jp/csl/ccrg/members/{yokoo, yuko, matubara}

Abstract

This paper presents a method for designing bundles in a combinatorial auction protocol that is robust against false-name bids. Internet auctions have become an integral part of Electronic Commerce and a promising field for applying AI technologies. However, the possibility of a new type of cheating called a false-name bid, i.e., a bid submitted under a fictitious name, has been pointed out.

A protocol called Leveled Division Set (LDS) protocol that is robust against false-name bids has been developed. However, this protocol requires the auctioneer to define a leveled division set. A leveled division set is a series of division sets, where a division set is a set of divisions and a division is a combination of bundles of goods. We need to solve a very complicated optimization problem to construct a leveled division set in order to obtain a good social surplus.

We have developed a heuristic method for overcoming this problem. In this method, we first find a good division with a winner determination algorithm, and then construct a leveled division set by using this division as a seed. Through a simulation, we show that our method can obtain a social surplus that is very close to optimal.

1 Introduction

Internet auctions have become an especially popular part of Electronic Commerce (EC). Various theoretical and practical studies on Internet auctions have already been conducted [Monderer and Tennenholtz, 1998; Wurman *et al.*, 1998]. Among these studies, those on combinatorial auctions have lately attracted considerable attention [Fujishima *et al.*, 1999; Leyton-Brown *et al.*, 2000; Klemperer, 1999; Sandholm, 1999; Lehmann *et al.*, 1999]. Although conventional auctions sell a single item at a time, combinatorial auctions sell multiple items with interdependent values simultaneously and allow the bidders to bid on any combination of items. In a combinatorial auction, a bidder can express complementary/substitutional preferences over multiple bids. By taking into account complementary/substitutional preferences, we can increase the participants' utilities and the revenue of the seller.

However, in [Sakurai *et al.*, 1999], the authors pointed out the possibility of a new type of cheating called *false-name bids*, i.e., an agent may try to profit from submitting false bids made under fictitious names, e.g., multiple e-mail addresses. Such a dishonest action is very difficult to detect since identifying each participant on the Internet is virtually impossible. Compared with collusion [Rasmusen, 1994; Klemperer, 1999], a false-name bid is easier to execute since it can be done by someone acting alone, while a bidder has to seek out and persuade other bidders to join in collusion.

The results reported in [Sakurai *et al.*, 1999; Yokoo *et al.*, 2000a] can be summarized as follows.

- The generalized Vickrey auction protocol (GVA) [Varian, 1995] is not robust against false-name bids in combinatorial auctions.
- There exists no combinatorial auction protocol that simultaneously satisfies incentive compatibility, Pareto efficiency, and individual rationality for all cases if agents can submit false-name bids.

In this paper, we concentrate on private value auctions [Mas-Colell *et al.*, 1995]. In private value auctions, each agent knows its own evaluation values of goods, which are independent of the other agents' evaluation values. We define an agent's utility as the difference between the true evaluation value of the allocated goods and the payment for the allocated goods. Such a utility is called a *quasi-linear* utility [Mas-Colell *et al.*, 1995]. These assumptions are commonly used for making theoretical analyses tractable.

In a traditional definition [Mas-Colell *et al.*, 1995], an auction protocol is (dominant strategy) incentive compatible, if bidding the true private values of goods is the dominant strategy for each agent, i.e., the optimal strategy regardless of the actions of other agents. The revelation principle states that in the design of an auction protocol we can restrict our attention to incentive compatible protocols without loss of generality [Mas-Colell *et al.*, 1995; Yokoo *et al.*, 2000a]. In other words, if a certain property (e.g., Pareto efficiency) can be achieved using some auction protocol in a dominant strategy equilibrium, i.e., the combination of dominant strategies of agents, the property can also be achieved using an incentive compatible auction protocol.

In this paper, we extend the traditional definition of incentive-compatibility so that it can address false-name bid manipulations, i.e., we define that an auction protocol is (dominant strategy) incentive compatible, if bidding the true private values of goods by using the true identifier is the dominant strategy for each agent. Also, we say that auction protocols are robust against false-name bids if each agent cannot obtain additional profit by submitting false-name bids. If such robustness is not satisfied, the auction protocol lacks incentive compatibility.

We say an auction protocol is Pareto efficient when the sum of all participants' utilities (including that of the auctioneer), i.e., the social surplus, is maximized in a dominant strategy equilibrium. In a more general setting, Pareto efficiency does not necessarily mean maximizing the social surplus. In an auction setting, however, agents can transfer money among themselves, and the utility of each agent is quasi-linear; thus the sum of the utilities is always maximized in a Pareto efficient allocation.

An auction protocol is individually rational if no participant suffers any loss in a dominant strategy equilibrium, i.e., the payment never exceeds the evaluation value of the obtained goods. In a private value auction, individual rationality is indispensable, i.e., no agent wants to participate in an auction where it might be charged more money than it is willing to pay.

The GVA protocol [Varian, 1995] is one instance of the well-known Clarke mechanism [Mas-Colell *et al.*, 1995]. It satisfies incentive compatibility, Pareto efficiency, and individual rationality in combinatorial auctions when there exists no false-name bid; however, this protocol is not robust against false-name bids [Sakurai *et al.*, 1999].

In [Yokoo *et al.*, 2000b], we developed a combinatorial auction protocol called the Leveled Division Set (LDS) protocol. This protocol satisfies incentive compatibility and individual rationality even if agents can submit false-name bids. This protocol does not satisfy Pareto efficiency, but it can achieve a better social surplus than that obtained by a trivial protocol that always sells goods in one bundle.

However, the LDS protocol requires the auctioneer to design a leveled division set, which is a series of division sets. A division set is a set of divisions and a division is a combination of bundles of goods. The auctioneer sells goods using these bundles. The obtained social surplus or the revenue of the auctioneer can vary significantly depending on the selection of the leveled division set. Although we presented the conditions that the leveled division set must satisfy to guarantee incentive compatibility [Yokoo *et al.*, 2000b], how to construct a leveled division set that could obtain a good social surplus is still an open question.

We need to solve a very complicated optimization problem in order to construct a leveled division set in such a way that a good social surplus can be obtained. This problem is much more complicated than winner determination problems [Fujishima *et al.*, 1999; Sandholm, 1999]. In a winner determination problem, given a set of bids, we try to find a combination of bids that maximizes the sum of the values of the bids. On the other hand, to construct a leveled division set, we need some model of the possible bids, e.g., a probability distribution of the values of the bids. We try to maximize the expected social surplus based on such a model. Also, a leveled division set is a series of division sets, where each division is a combination of bundles. Finding an optimal solution for this problem becomes infeasible when the number of goods is large.

In this paper, we develop a heuristic method to construct a good leveled division set. This method uses a winner determination algorithm as a building block, i.e., this method first finds a good division using a winner determination algorithm, and then constructs a leveled division set using this division as a seed.

In the following, we first describe the LDS protocol. Then, we show the method for constructing a good leveled division set. Finally, we show evaluation results through a simulation.

2 LDS Protocol

In the following, we define several terms and notations. To help readability, we use three different types of parentheses to represent sets: (), $\{\}$, and [].

- A set of agents $N = \{1, 2, \ldots, n\}$
- A set of all auctioned goods $M = (1, 2, \ldots, m)$
- A division, which is a set of bundles, $D = \{S \subseteq M \mid S \cap S' = \emptyset \text{ for every } S, S' \in D\}$[1]
- For each good j, the reservation price r_j is defined.
- For a bundle S, we define $R(S)$ as $\sum_{j \in S} r_j$.

A *leveled division set* is defined as follows:

- Levels are defined as $1, 2, \ldots, max\_level$.
- For each level i, a division set $SD_i = [D_{i1}, D_{i2}, \ldots]$ is defined.

A leveled division set must satisfy the following three conditions.

- $SD_1 = [\{M\}]$ — the division set of level 1 contains only one division, which consists of a bundle of all goods.
- For each level and its division set, a union of multiple bundles in a division is always included in a division of a smaller level, i.e., $\forall i \geq 2, \forall D_{ik} \in SD_i, \forall D' \subseteq D_{ik}$, where $|D'| \geq 2, S_u = \bigcup_{S \in D'} S$, there exists a level $j < i$, with a division set SD_j, where $D_{jl} \in SD_j$ and $S_u \in D_{jl}$.
- For each level and its division set, each bundle in a division is not included in a division of a different level[2], i.e., $\forall i, \forall D_{ik} \in SD_i, \forall S \in D_{ik}, \forall j \neq i, \forall D_{jl} \in SD_j, S \notin D_{jl}$.

Figure 1 shows examples of leveled division sets. Case 1 shows one instance where there are two goods (A and B), and case 2 and case 3 show instances where there are three and four goods, respectively.

[1] Note that we don't require that $\bigcup_{S \in D} S = M$ holds, i.e., satisfying $\bigcup_{S \in D} S \subseteq M$ is sufficient.

[2] This condition is not contradictory to the second condition, since the second condition involves the union of multiple bundles.

	case 1	case 2	case 3
level 1	[{(A,B)}]	[{(A,B,C)}]	[{(A,B,C,D)}]
level 2	[{(A),(B)}]	[{(A,B)}, {(B,C)}, {(A,C)}]	[{(A,B,C)}, {(B,C,D)}, {(A,D)}]
level 3		[{(A),(B),(C)}]	[{(A),(D),(B,C)}]

Figure 1: Example of Leveled Division Sets

To execute the LDS protocol, the auctioneer must predefine the leveled division set and the reservation prices of goods. Each agent x declares its evaluation value $B(x, S)$ for each bundle S, which may or may not be true. We show a detailed description of the LDS protocol in the appendix. The outline of the protocol is as follows. The LDS protocol first tries to sell goods using the division in level 1, i.e., if there exists an evaluation value that is larger than $R(M)$, the goods are sold in one bundle. Otherwise, the protocol tries to sell goods using level 2 divisions, and so on. We say that the *applied level* of the LDS protocol is i if the goods are sold using a division in level i.

An intuitive explanation of why the LDS protocol satisfies incentive compatibility is as follows. Let us assume that agent x uses two false names x' and x'' to obtain bundles $S_{x'}$ and $S_{x''}$, respectively, at level i. Now, let us assume that agent x declares the evaluation value $R(S)$ for the bundle $S = S_{x'} \cup S_{x''}$ by using a single identifier. From the condition of a leveled division set, there exists a level $j < i$, where $S \in D_{jl}$, $D_{jl} \in SD_j$ holds. In this case, the condition in Step 1 of LDS(j) is satisfied, i.e., only agent x declares evaluation values that are larger than or equal to the sum of reservation prices. Thus, agent x can obtain S by paying the minimal price $R(S)$. Therefore, by using a single identifier, the payment of agent x becomes smaller than (or equal to) the payment when agent x uses two false names.

3 Constructing Leveled Division Set

3.1 Basic Concepts

We assume that the auctioneer knows the probability distribution of the possible evaluation values of participants. The goal is to find a leveled division set and the reservation prices of goods so that the expected social surplus can be maximized.

For the number of goods m, the number of possible bundles of goods is 2^m. A division is a set of bundles and a leveled division set is a series of division sets. Clearly, enumerating all possible leveled division sets and finding the optimal one becomes infeasible when the number of goods is large. Even if we use a best-first search algorithm such as A*, it is difficult to find an optimal leveled division without a very accurate function that estimates the expected social surplus for a partially defined leveled division set. Furthermore, we need to optimize the reservation prices of goods as well as the leveled division set.

In this paper, we make the following assumptions to attack this problem by a heuristic method.

- As in winner determination problems [Fujishima *et al.*, 1999; Sandholm, 1999], the number of bundles that participants actually need is relatively small compared with the number of all possible bundles 2^m (sparseness of bids).
- The auctioneer knows the probability distributions of the highest evaluation values of these bundles[3]. Also, these probabilities are mutually independent.

With these assumptions, we can find a candidate for a division that can obtain a good social surplus using a winner determination algorithm [Fujishima *et al.*, 1999; Sandholm, 1999]. More specifically, instead of the actual value of a bid for each bundle, we use the expected value of the highest evaluation value for each bundle to solve the winner determination problem and obtain a division (we call this division *goal division*). If we can sell goods using this goal division, the obtained social surplus would be relatively good on average.

However, to sell goods using this goal division, we need to include additional divisions in earlier levels to satisfy the conditions of the leveled division set described in the previous section. Therefore, we put the goal division at level 3, and put additional divisions at level 2. By setting the reservation price of each good in an appropriate range, we can expect that goods will not be sold at level 1 or level 2, and we can sell goods using the goal division.

For example, let us assume that there are five goods A, B, C, D and E. Also, let us assume that by running a winner determination algorithm using the average value of each bundle, we obtain a goal division $\{(A), (B), (C), (D, E)\}$. Then, we set level 3 of the leveled division set as $[\{(A), (B), (C), (D, E)\}]$.

To satisfy the conditions of the leveled division set, we must put all unions of multiple bundles in this goal division at level 1 or level 2. There are four bundles in this goal division, so we must create unions of two bundles and three bundles and put them in the divisions at level 2. If two newly created bundles, each of which is a union of two bundles, do not have any goods in common, we can construct a division using these bundles. For example, we can put divisions $\{(A, B), (C, D, E)\}$,$\{(A, C), (B, D, E)\}$ and $\{(A, D, E), (B, C)\}$ in level 2. Then, all unions of two bundles appear in level 2. For a bundle that is a union of three bundles, we create a division that consists of only this bundle, and add them to level 2, e.g., we add $\{(A, B, C)\}$, $\{(A, B, D, E)\}$,$\{(A, C, D, E)\}$, and $\{(B, C, D, E)\}$.

Also, we can add a new goal division in level 3, as long as the division does not contain a bundle that appears in level 2. For example, we can add a division

[3] If we assume that the auctioneer has knowledge of the probability distributions of the evaluation values of agents, the probability distributions of the highest evaluation values can be calculated based on this knowledge. Since the LDS protocol is incentive compatible, we can assume that agents truthfully bid their true evaluation values.

$\{(A),(B),(C,D),(E)\}$, since it does not contain any bundles in level 2. To add this division in level 3, we must add divisions $\{(A,B,C,D)\}$,$\{(A,B,E)\}$,$\{(A,C,D),(B,E)\}$, and $\{(A,E),(B,C,D)\}$ in level 2.

It must be noted that even if the above assumptions do not hold, i.e., there might be unexpected bids or bids might be correlated, the LDS protocol using the leveled division set obtained by this method satisfies incentive compatibility (and the robustness against false-name bids). The violations of the above assumptions might degrade the obtained social surplus, but the protocol satisfies incentive compatibility as long as the required conditions of the leveled division set are satisfied.

3.2 Details of Method for Constructing Leveled Division Set

We show the details of the method for constructing a leveled division set. Let us represent the set of bundles that participants actually need as BS. For each bundle $S \in BS$, we assume that the auctioneer knows the probability distribution of the the highest evaluation value $B_{max}(S)$. Let us represent the expected value of $B_{max}(S)$ as $E(B_{max}(S))$. To help readability, for a division $D = \{S_1, S_2, \ldots\}$, we represent a set of unions of k elements of D as $U(D,k)$.

We execute the following procedure. In the initial state, the level 1 division set SD_1 is $[\{M\}]$, and the level 2 and level 3 division sets SD_2 and SD_3 are empty sets $[]$.

Step 1: Find a goal division D that maximizes $\sum_{S \in D} E(B_{max}(S))$ using a winner determination algorithm[4]. Add D to SD_3. We exclude a division that is already an element of SD_3. Also, we exclude a division D where any element of $U(D,k)$ for all $k \geq 2$ is also an element of BS. This is because we need to put each element of $U(D,k)$ at level 2. Therefore, if any element is also a member of BS, there might be a relatively high evaluation value for this element and we would need to sell goods using the level 2 division. Also, we exclude a division D that contains a bundle that is already included in level 2. This is because we cannot put such D at level 3 by the conditions of the leveled division set.

Step 2: For all $D' = \{S'\}$, where $t = |D|$, $S' \in U(D, t-1)$, we add D' to SD_2. Furthermore, for all $D'' = \{S_1, S_2\}$, where $S_1 \in U(D,u)$, $S_2 \in U(D, t-u)$, $u \geq 2$ and $S_1 \cap S_2 = \emptyset$, we add D'' to SD_2.

Step 3: When adding another goal division, go to Step 1, otherwise, terminate the procedure.

The appropriate number of goal divisions in level 3 varies depending on the problem setting. In the problem settings used in the next section, we found that improvement of the obtained social surplus was saturated after adding about 20 goal divisions.

For solving a winner determination problem, we don't have to find the optimal solution. Since we use the expected value instead of the actual value, we cannot say that the optimal solution for the winner determination problem always gives the best goal division. Therefore, using an approximate algorithm (such as [Sakurai *et al.*, 2000]) would be sufficient.

[4]For simplicity, we assume $|D| \geq 3$ and $\bigcup_{S \in D} S = M$ hold. If D is a set of two bundles, we can put D into SD_2. Also, if some goods are not included in D, we must add these goods to D.

4 Evaluations

In this section, we show simulation results that demonstrate the quality of the obtained leveled division set using our heuristic method. We determine a bundle S that participants actually need and the probability distribution of the highest evaluation value $B_{max}(S)$ by the following method.

- For each bundle S, we determine the number of goods in S (represented as $|S|$) by using a binomial distribution $B(p,m)$, i.e., the probability that the number of goods is j is given by $p^j(1-p)^{m-j}m!/(j!(m-j)!)$ [Fujishima *et al.*, 1999]. Next, we randomly choose $|S|$ goods included in S. Also, we choose $E(B_{max}(S))$ randomly from within the range of $[(1-\epsilon)|S|, (1+\epsilon)|S|]$. Then, we set the distribution of $B_{max}(S_k)$ as a uniform distribution $[(1-q)E(B_{max}(S)), (1+q)E(B_{max}(S))]$, where $0 \leq q \leq 1$.

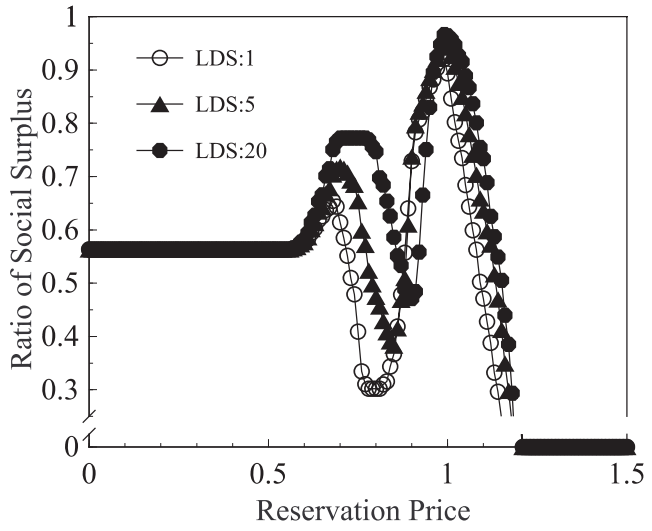
If q is very small, the auctioneer has very good knowledge of the highest evaluation value of S. If q is relatively large compared with ϵ, the auctioneer has little knowledge of the possible highest evaluation value of S, and the probability that the goal division can actually achieve a Pareto efficient social surplus becomes low.

Figure 2 (a) shows the average ratio of the obtained social surplus to the Pareto efficient social surplus by varying the reservation price[5]. In Figure 2, we set $m = 10$, the number of bundles $|BS| = 200$, $p = 0.2$, $\epsilon = 0.1$, and $q = 0.1$. Each data point is the average of 100 problem instances, in which the evaluation values of bundles are generated based on the probability distribution determined by the above method. In Figure 2 (a), we plot the results where we put only one goal division (LDS:1), 5 goal divisions (LDS:5), and 20 goal divisions (LDS:20). Also, Figure 2 (b) shows the trends of how the applied levels change according to the reservation price in LDS:20. More specifically, for each value of the reservation price, we show the cumulative frequency of three levels in which the goods are sold,

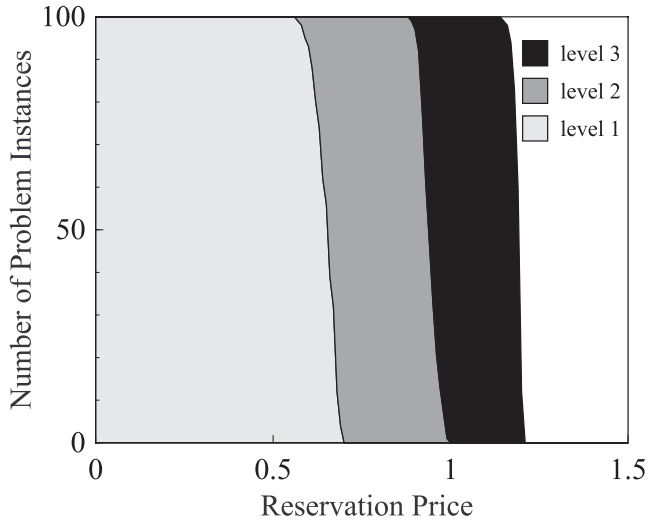
As shown in Figure 2 (b), when the reservation price is low, the applied level is 1, i.e., the LDS protocol sells goods using the level 1 division, which means that the protocol always sell goods in one bundle to a single agent. When the reservation price is around 0.7, the applied level becomes 2. Also, the social surplus drops when the reservation price is around 0.9. This is because only one bundle in a level 2 division (which consists of two bundles) is sold in some problem instances.

If we set the reservation price in an appropriate range (in this case, around 1.0), the applied level becomes 3 and the protocol can sell goods using the goal divisions; thus the obtained social surplus becomes very good (more than 90% of the optimal) even if we put only one division at level 3. By

[5]We use an identical reservation price for all goods since in current problem settings, the evaluation values of goods are basically similar. If the evaluation values of goods can be very different, we need to set different reservation prices for different goods.



(a) Obtained Social Surplus



(b) Applied Levels (LDS:20)

Figure 2: Evaluation Results (m=10, q=0.1)

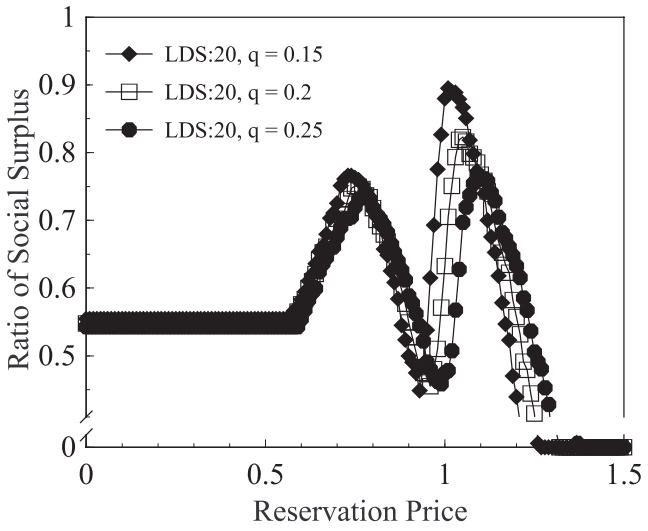


Figure 3: Obtained Social Surplus (m=10, q=0.15, 0.2, 0.25)

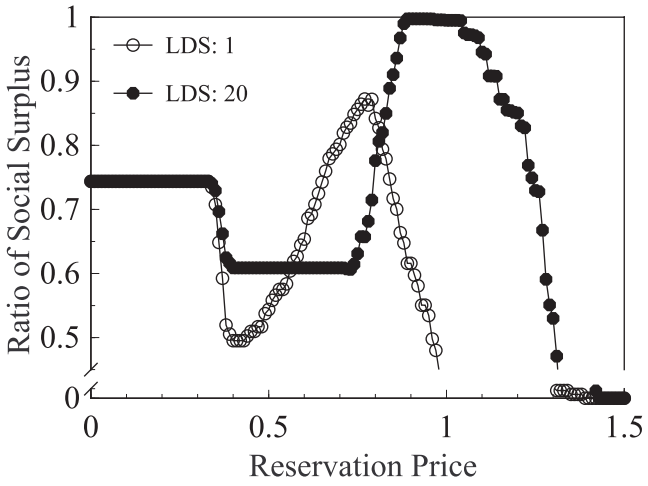


Figure 4: Obtained Social Surplus (m=200, q=0.3)

putting 20 goal divisions at level 3, the obtained social surplus becomes very close to the optimal. We found the improvement of the obtained social surplus becomes saturated after adding 20 goal divisions.

In Figure 3, we show the results of LDS:20, where we set $q = 0.15, 0.2$, and 0.25. As expected, when q becomes relatively large compared with $\epsilon = 0.1$, the probability that the goal division can actually obtain a Pareto efficient social surplus becomes very low and the obtained social surplus of the LDS protocol becomes low. However, even when $q = 0.2$, the LDS protocol can obtain a social surplus that is around 82% of the optimal.

Figure 4 shows the results obtained when the number of goods becomes large, i.e., we set $m = 200$, $|BS| = 1000$, $p = 0.2$, $\epsilon = 0.2$, and $q = 0.3$. Each data point is the average of 50 problem instances. We can see that even though q is relatively large, the obtained social surplus is very close to the optimal. In this case, the sizes of bundles in the goal division tend to be large, e.g., a bundle of 40 goods. Therefore, the possibility that goods are sold by level 2 divisions, which contain very large bundles (e.g., 80 goods), becomes very small. Thus, we can easily set the reservation price so that the applied level becomes 3.

Figure 5 shows the results obtained when adding some noise to problem instances. More specifically, we add several unexpected bids, i.e., the bids submitted by agents whose evaluation values do not correspond to the model the auctioneer has. A noise bid is generated using basically the same method for generating original bids. In Figure 5, the x-axis shows the number of additional noise bids. The parameter settings are identical to those of Figure 4, and we show the results of LDS:20, where the reservation price is 1. We can

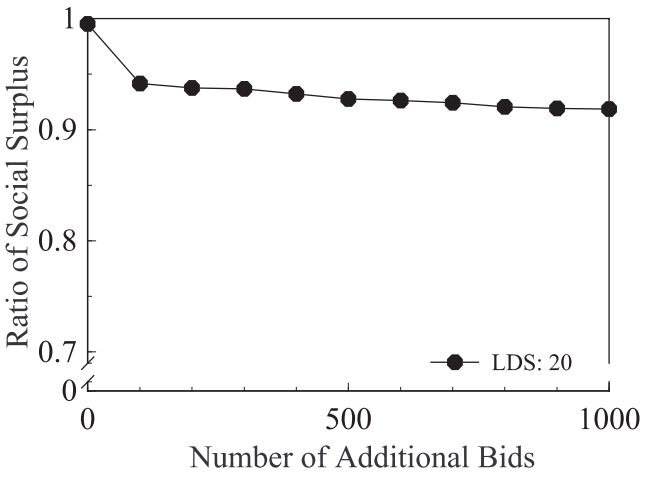


Figure 5: Effect of Adding Noise

see that our method is very robust against additional noise bids. Even if we add 1000 noise bids, which is equal to $|BS|$, the obtained social surplus is around 92% of the optimal.

5 Conclusions

In this paper, we developed a heuristic method for constructing a good division set that can be used in the LDS protocol. This method uses a winner determination algorithm to find a good division, and then constructs a leveled division set by using this division as a seed. Evaluation results showed that the LDS protocol using the leveled division set constructed by our method can obtain a social surplus that is very close to optimal when the auctioneer has reasonably good knowledge of the highest evaluation values of agents. Furthermore, the protocol can still obtain a good social surplus when the knowledge of the auctioneer is relatively poor. Our future works include evaluating this method in more realistic problem settings based on real application problems such as [Leyton-Brown *et al.*, 2000], and extending a double auction protocol that is robust against false-name bids [Yokoo *et al.*, 2001] so that it can handle multiple items.

References

[Fujishima *et al.*, 1999] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computation complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 548–553, 1999.

[Klemperer, 1999] Paul Klemperer. Auction theory: A guide to the literature. *Journal of Economics Surveys*, 13(3):227–286, 1999.

[Lehmann *et al.*, 1999] Daniel Lehmann, Liadan Ita O'Callaghan, and Yoav Shoham. Truth revelation in approximately efficient combinatorial auction. In *Proceedings of the First ACM Conference on Electronic Commerce (EC-99)*, pages 96–102, 1999.

[Leyton-Brown *et al.*, 2000] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the Second ACM Conference on Electronic Commerce (EC-00)*, pages 66–76, 2000.

[Mas-Colell *et al.*, 1995] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[Monderer and Tennenholtz, 1998] Dov Monderer and Moshe Tennenholtz. Optimal auctions revisited. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 32–37, 1998.

[Rasmusen, 1994] Eric Rasmusen. *Games and Information*. Blackwell, 1994.

[Sakurai *et al.*, 1999] Yuko Sakurai, Makoto Yokoo, and Shigeo Matsubara. A limitation of the Generalized Vickrey Auction in Electronic Commerce: Robustness against false-name bids. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 86–92, 1999.

[Sakurai *et al.*, 2000] Yuko Sakurai, Makoto Yokoo, and Koji Kamei. An efficient approximate algorithm for winner determination in combinatorial auctions. In *Proceedings of the Second ACM Conference on Electronic Commerce (EC-00)*, pages 30–37, 2000.

[Sandholm, 1999] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auction. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 542–547, 1999.

[Varian, 1995] Hal R. Varian. Economic mechanism design for computerized agents. In *Proceedings of the First Usenix Workshop on Electronic Commerce*, 1995.

[Wurman *et al.*, 1998] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Proceedings of the Second International Conference on Autonomous Agents (Agents-98)*, pages 301–308, 1998.

[Yokoo *et al.*, 2000a] Makoto Yokoo, Yuko Sakurai, and Shigeo Matsubara. The effect of false-name declarations in mechanism design: Towards collective decision making on the Internet. In *Proceedings of the Twentieth International Conference on Distributed Computing Systems (ICDCS-2000)*, pages 146–153, 2000.

[Yokoo *et al.*, 2000b] Makoto Yokoo, Yuko Sakurai, and Shigeo Matsubara. Robust combinatorial auction protocol against false-name bids. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 110–115, 2000.

[Yokoo *et al.*, 2001] Makoto Yokoo, Yuko Sakurai, and Shigeo Matsubara. Robust double auction protocol against false-name bids. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-2001)*, 2001. (to appear).

Appendix: Details of LDS Protocol

For a division $D = \{S_1, S_2, \ldots\}$ and one possible allocation of goods G, we say G is allowed under D if each bundle in D is allocated to different agents in G. Also, we allow that some bundle is not allocated to any agent. In that case, we assume that the bundle is allocated to a dummy agent d, whose evaluation value of the bundle S is equal to $R(S)$. For each level i and its division set $SD_i = [D_{i1}, D_{i2}, \ldots]$, we represent a union of all allowed allocations for each element of SD_i as SG_i. The declared evaluation value of agent x for an allocation G (represented as $v_x(G)$) is defined as $B(x, S)$ if S is allocated to agent x in G, otherwise $v_x(G) = 0$. Also, we define the evaluation value of a dummy agent d for an allocation G as the sum of the reservation prices of goods that are not allocated to real agents in G.

The winners and payments are determined by calling the procedure LDS(1). LDS(i) is a recursive procedure defined as follows. Note that the procedures in GVA(i) are equivalent to those in the GVA [Varian, 1995], except that the possible allocations are restricted to SG_i.

Procedure LDS(i)

Step 1: If there exists only one agent $x \in N$ whose evaluation values satisfy the following condition: $\exists D_{ik} \in SD_i, \exists S_x \in D_{ik}$, where $B(x, S_x) \geq R(S_x)$, then compare the results obtained by the procedure GVA(i) and LDS$(i+1)$, and choose the one that gives the larger utility for agent x. If the condition of Step 1 is also satisfied for LDS$(i+1)$, then also compare with the results of GVA$(i+1)$ and LDS$(i+2)$, and so on. When choosing the result of LDS$(i+1)$, we don't assign any good, nor transfer money, to agents other than x, although the assigned goods for agent x and its payment are calculated as if goods were allocated to the other agents.

Step 2: If there exist at least two agents $x_1, x_2 \in N, x_1 \neq x_2$ whose evaluation values satisfy the following condition: $\exists D_{ik} \in SD_i, \exists D_{il} \in SD_i, \exists S_{x_1} \in D_{ik}, \exists S_{x_2} \in D_{il}$, where $B(x_1, S_{x_1}) \geq R(S_{x_1})$, $B(x_2, S_{x_2}) \geq R(S_{x_2})$, then apply the procedure GVA(i).

Step 3: Otherwise: call LDS$(i+1)$, or terminate if $i = max\_level$.

Procedure GVA(i)**:** Choose an allocation $G^* \in SG_i$ such that it maximizes $\sum_{y \in N \cup \{d\}} v_y(G)$. The payment of agent x (represented as p_x) is calculated as $\sum_{y \neq x} v_y(G^*_{\sim x}) - \sum_{y \neq x} v_y(G^*)$, where $G^*_{\sim x} \in SG_i$ is the allocation that maximizes the sum of all agents' (including the dummy agent d) evaluation values except that of agent x.

CABOB: A Fast Optimal Algorithm for Combinatorial Auctions*

Tuomas Sandholm
sandholm@cs.cmu.edu
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Subhash Suri
suri@cs.ucsb.edu
Department of Computer Science
University of California
Santa Barbara, CA 93106

Andrew Gilpin **David Levine**
{agilpin,dlevine}@CombineNet.com
CombineNet, Inc.
311 S. Craig St
Pittsburgh, PA 15213

Abstract

Combinatorial auctions where bidders can bid on bundles of items can lead to more economical allocations, but determining the winners is $\mathcal{NP}$-complete and inapproximable. We present CABOB, a sophisticated search algorithm for the problem. It uses decomposition techniques, upper and lower bounding (also across components), elaborate and dynamically chosen bid ordering heuristics, and a host of structural observations. Experiments against CPLEX 7.0 show that CABOB is usually faster, never drastically slower, and in many cases drastically faster. We also uncover interesting aspects of the problem itself. First, the problems with short bids that were hard for the first-generation of specialized algorithms are easy. Second, almost all of the CATS distributions are easy, and become easier with more bids. Third, we test a number of random restart strategies, and show that they do not help on this problem because the run-time distribution does not have a heavy tail (at least not for CABOB).

1 Introduction

Auctions are important mechanisms for resource and task allocation in multiagent systems. In many auctions, a bidder's valuation for a combination of distinguishable items is not the sum of the individual items' valuations—it can be more or less. *Combinatorial auctions (CAs)* where bidders can bid on bundles of items allow bidders to express *complementarity* (and, with a rich enough bidding language, also *substitutability* among the items [Sandholm, 1999; Fujishima *et al.*, 1999; Sandholm, 2000; Nisan, 2000]). This expressiveness can lead to more economical allocations of the items because bidders do not get stuck with partial bundles of low value. This has been demonstrated, for example, in airport landing slot allocation [Rassenti *et al.*, 1982], and in transportation exchanges [Sandholm, 1993].

However, determining the winners in a combinatorial auctions is computationally complex. There has recently been a surge of research into addressing that [Rothkopf *et al.*, 1998; Sandholm, 1999; Fujishima *et al.*, 1999; Lehmann *et al.*, 1999; Sandholm and Suri, 2000; Andersson *et al.*, 2000; Hoos and Boutilier, 2000; de Vries and Vohra, 2000]. In this paper we present a fast optimal search algorithm for the problem. Section 2 defines the problem. Our algorithm is presented in Section 3. Section 4 discusses bid ordering heuristics. Experimental results are presented in Sections 5–7. Random restart strategies are discussed in Section 8. Section 9 presents conclusions and future research directions.

*This work was funded by, and conducted at, CombineNet, Inc.

2 The Winner Determination Problem

In this section we define the winner determination problem.

Definition 1 *The auctioneer has a set of items, $M = \{1, 2, \ldots, m\}$, to sell, and the buyers submit a set of bids, $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$. A bid is a tuple $B_j = \langle S_j, p_j \rangle$, where $S_j \subseteq M$ is a set of items and $p_j \geq 0$ is a price. The* binary combinatorial auction winner determination problem *is to label the bids as winning or losing so as to maximize the auctioneer's revenue under the constraint that each item can be allocated to at most one bidder:*

$$\max \sum_{j=1}^{n} p_j x_j \quad \text{s.t.} \quad \sum_{j|i \in S_j} x_j \leq 1,\ i \in \{1..m\}$$
$$x_j \in \{0,1\}$$

This is $\mathcal{NP}$-complete [Rothkopf *et al.*, 1998], and it cannot even be approximated to a ratio of $n^{1-\epsilon}$ in polynomial time (unless $\mathcal{P} = \mathcal{NP}$) [Sandholm, 1999].

If bids could be accepted partially, the problem would become a linear program (LP) which can be solved in polynomial time. Here we present the LP-formulation and its dual because we will use them in several ways in our algorithm.

$$\begin{array}{ll} LP & DUAL \\ \max \sum_{j=1}^{n} p_j x_j & \min \sum_{i=1}^{m} y_i \\ \sum_{j|i \in S_j} x_j \leq 1,\ i \in \{1..m\} & \sum_{i \in S_j} y_i \geq p_j,\ j \in \{1..n\} \\ x_j \in \Re & y_i \in \Re \end{array}$$

In this continuous setting, the *shadow price* y_i gives the price for each individual item i.[1] In the binary case individual items cannot generally be given prices, but each y_i value from DUAL gives an upper bound on the price of item i.

[1] If there are some items that are not included in any bids, we have to add to the DUAL the constraint $y_i \geq 0$ for those items. Alternatively (and preferably), such items can simply be removed as a preprocessing step.

3 Description of the Algorithm

Our algorithm, *CABOB (Combinatorial Auction Branch On Bids)*, is a tree search algorithm that branches on bids. The high-level idea of branching on bids was already proposed by [Sandholm and Suri, 2000] in the BOB algorithm. However, BOB was not implemented. CABOB incorporates many of the techniques proposed in BOB and a host of additional ones. All of them have been implemented.

The skeleton of CABOB is a depth-first branch-and-bound tree search that branches on bids. The value of the best solution found so far is a global variable $\tilde{f}^*$. Initially, $\tilde{f}^* = 0$.

A data structure called the *bid graph* is maintained. We denote it by G. The nodes of the graph correspond to bids that are still available to be appended to the search path, i.e., bids that do not include any items that have already been allocated. Two vertices in G share an edge whenever the corresponding bids share items.[2] As vertices are removed from G when going down a search path, the edges that they are connected to are also removed. As vertices are re-inserted into G when backtracking, the edges are also reinserted.

The following pseudocode of CABOB makes calls to several special cases which will be introduced later. For readability, we omit how the solution (set of winning bids) is updated in conjunction with every update of $\tilde{f}^*$.

As will be discussed later, we use a technique for pruning across independent subproblems (components of G). To support this, we use a parameter, MIN, to denote the minimum revenue that the call to CABOB must return (not including the revenue from the path so far or from neighbor components) to be competitive with the best solution found so far. The revenue from the bids that are winning on the search path so far is called g. It includes the lower bounds (or actual values) of neighbor components of each search node on the path so far.

The search is invoked by calling $CABOB(G, 0, 0)$.

Algorithm 3.1 $CABOB(G, g, MIN)$

1. *Apply special cases COMPLETE and NO_EDGES*
2. *Run DFS on G; let c be number of components found, and let $G_1, G_2, ..., G_c$ be the c independent bid graphs*
3. *Calculate an upper bound U_i for each component i*
4. *If $\sum_{i=1}^{c} U_i \leq MIN$, then return* 0
5. *Apply special case INTEGER*
6. *Calculate a lower bound L_i for each component i*
7. $\Delta \leftarrow g + \sum_{i=1}^{c} L_i - \tilde{f}^*$
8. *If* $\Delta > 0$, *then*
   $\tilde{f}^* \leftarrow \tilde{f}^* + \Delta$
   $MIN \leftarrow MIN + \Delta$
9. *If* $c > 1$ *then goto (11)*
10. *Choose next bid B_k to branch on (use articulating bids first if any)*
    10.a. $G \leftarrow G - \{B_k\}$
    10.b. For all B_j s.t. $B_j \neq B_k$ and $S_j \cap S_k \neq \emptyset$, $G \leftarrow G - \{B_j\}$
    10.c. $\tilde{f}^*_{old} \leftarrow \tilde{f}^*$
    10.d. $f_{in} \leftarrow CABOB(G, g + p_k, MIN - p_k)$
    10.e. $MIN \leftarrow MIN + (\tilde{f}^* - \tilde{f}^*_{old})$
    10.f. For all B_j s.t. $B_j \neq B_k$ and $S_j \cap S_k \neq \emptyset$, $G \leftarrow G \cup \{B_j\}$
    10.g. $\tilde{f}^*_{old} \leftarrow \tilde{f}^*$
    10.h. $f_{out} \leftarrow CABOB(G, g, MIN)$
    10.i. $MIN \leftarrow MIN + (\tilde{f}^* - \tilde{f}^*_{old})$
    10.j. $G \leftarrow G \cup \{B_k\}$
    10.k. Return $\max\{f_{in}, f_{out}\}$
11. $F^*_{solved} \leftarrow 0$
12. $H_{unsolved} \leftarrow \sum_{i=1}^{c} U_i, \quad L_{unsolved} \leftarrow \sum_{i=1}^{c} L_i$
13. *For each component* $i \in \{1, \ldots, c\}$ *do*
    13.a. If $F^*_{solved} + H_{unsolved} \leq MIN$, *return* 0
    13.b. $g'_i \leftarrow F^*_{solved} + (L_{unsolved} - L_i)$
    13.c. $\tilde{f}^*_{old} \leftarrow \tilde{f}^*$
    13.d. $f^*_i \leftarrow CABOB(G_i, g + g'_i, MIN - g'_i)$
    13.e. $MIN \leftarrow MIN + (\tilde{f}^* - \tilde{f}^*_{old})$
    13.f. $F^*_{solved} \leftarrow F^*_{solved} + f^*_i$
    13.g. $H_{unsolved} \leftarrow H_{unsolved} - U_i$
    13.h. $L_{unsolved} \leftarrow L_{unsolved} - L_i$
14. *Return* F^*_{solved}

We now discuss the techniques of CABOB in more length.

Upper Bounding

In step (3), CABOB uses an upper bound on the revenue the unallocated items can contribute. If the current solution cannot be extended to a new optimal solution under the optimistic assumption that the upper bound is met, CABOB prunes the search path.

Any technique for devising an upper bound could be used here. We solve the remaining LP, whose objective function value gives an upper bound. CABOB does not make copies of the LP table, but incrementally adds (deletes) rows from the LP table as bids are removed (re-inserted) into G as the search proceeds down a path (backtracks). Also, as CABOB moves down a search path, it remembers the LP solution from the parent and uses it as a starting solution for the child's LP.

It is not always necessary to run the LP to optimality. Before starting the LP, one could look at the condition in step (4) to determine the minimum revenue the LP has to produce so that the search branch would not be pruned.[3] Once the LP solver finds a solution that exceeds the threshold, it could be stopped without pruning the search branch. If the LP solver does not find a solution that exceeds the threshold and runs to completion, the branch could be pruned. However, CABOB always runs the LP to completion since it uses the solutions from the LP and the DUAL in several ways.

[2] Since G can be constructed incrementally as bids are submitted, its construction does not add to winner determination time after the auction closes. Therefore, in the experiments, the time to construct G is not included (in almost all cases it was negligible anyway, but for instances with very long bids it sometimes took almost as much time as the search).

[3] In the case of multiple components, when determining how high a revenue one component's LP has to return, the exact solution values from solved neighbor components would be included, as well as the upper bounds from the unsolved neighbor components.

Our experiments showed that the upper bound from LP is significantly tighter than those proposed for previous combinatorial auction winner determination algorithms [Sandholm, 1999; Fujishima *et al.*, 1999; Sandholm and Suri, 2000]. The time taken to solve the LP at every node is negligible compared to the savings in search due to enhanced pruning.

The INTEGER special case

If the LP happens to return integer values ($x_j = 0$ or $x_j = 1$) for each bid j (this occurs more often than we expected), CABOB makes the bids with $x_j = 1$ winning, and those with $x_j = 0$ losing. This is clearly an optimal solution for the remaining bids. CABOB updates $\tilde{f}^*$ if the solution is better than the best so far. CABOB then returns from the call without searching further under that node.

It is easy to show that if some of the x_j values are not integer, we cannot simply accept the bids with $x_j = 1$. Neither can we simply reject the bids with $x_j = 0$. Either approach can compromise optimality.

Lower Bounding

In step (6), CABOB calculates a lower bound on the revenue that the remaining items can contribute. If the lower bound is high, it can allow $\tilde{f}^*$ to be updated, leading to more pruning and less search in the subtree rooted at that node.

Any lower bounding technique could be used here. We use the following rounding technique. In step (3), CABOB solves the remaining LP anyway, which gives an "acceptance level" $x_j \in [0, 1]$ for every remaining bid B_j. We insert all bids with $x_j > 0.5$ into the lower bound solution. We then try to insert the rest of the bids in decreasing order of x_j, skipping bids that share items with bids already in the lower bound. It is easy to prove that this method gives a lower bound.

While rounding techniques are known to provide reasonably good lower bounds on average, in the future we are planning to try other lower bounding techniques within CABOB such as stochastic local search [Hoos and Boutilier, 2000].

Exploiting decomposition

In step (2), CABOB runs an $O(|E| + |V|)$ time depth-first-search (DFS) in the bid graph G. Each tree in the depth-first forest is a connected component of G. Winner determination is then conducted in each component independently. Since search time is superlinear in the size of G, this decomposition leads to a time savings. The winners are determined by calling CABOB on each component separately. As the experiments show, this can lead to a drastic speedup.

Upper and lower bounding across components

In addition to regular upper and lower bounding, somewhat unintuitively, we can achieve further pruning, without compromising optimality, by exploiting information across the independent components. When starting to solve a component, CABOB checks how much that component would have to contribute to revenue in the context of what is already known about bids on the search path so far *and the neighboring components*. Specifically, when determining the MIN value for calling CABOB on a component, the revenue that the current call to CABOB has to produce (the current MIN value), is decremented by the revenues from solved neighbor components and the lower bounds from unsolved neighbor components. *Our use of a MIN value allows the algorithm to work correctly even if on a single search path there may be several search nodes where decomposition occurred, interleaved with search nodes where decomposition did not occur.*

Every time a better global solution is found and $\tilde{f}^*$ is updated, all MIN values in the search tree should be incremented by the amount of the improvement since now the bar of when search is useful has been raised. CABOB handles these updates without separately traversing the tree when an update occurs. CABOB directly updates MIN in step (8), and updates the MIN value of any parent node after the recursive call to CABOB returns.

CABOB also uses lower bounding across components. At any search node, the lower bound includes the revenues from the bids that are winning on the path, the revenues from the solved neighbor components of search nodes on the path, the lower bounds of the unsolved neighbor components of search nodes on the path, and the lower bound on the revenue that the unallocated items in the current search node can contribute.

Due to upper and lower bounding across components (and due to updating of $\tilde{f}^*$), the order of tackling the components can potentially make a difference in speed. CABOB currently tackles components in the order that they are found in the DFS. We plan to study more elaborate component ordering in future research.

Forcing a decomposition via articulation bids

In addition to checking whether a decomposition has occurred, CABOB strives for a decomposition. In the bid choice in step (10), it picks a bid that leads to a decomposition, if such a bid exists. Such bids whose deletion disconnects G are called *articulation bids*. Articulation bids are identified in $O(|E| + |V|)$ time by a slightly modified DFS in G, as proposed in [Sandholm and Suri, 2000].

The scheme of always branching on an articulation bid, if one exists, is often at odds with price-based bid ordering schemes, discussed later. As proved in [Sandholm and Suri, 2000], no scheme from the articulation-based family dominates any scheme from the price-based family, or vice versa, in general. However, our experiments showed that in practice it almost always pays off to branch on articulation bids if they exist (because decomposition reduces search drastically).

Even if a bid is not an articulation bid, and would not lead to a decomposition if the bid is assigned losing, it might lead to a decomposition if it is assigned winning because that removes the bid's neighbors from G as well. This is yet another reason to assign a bid that we branch on to be winning before assigning it to be losing (value ordering). Also, in bid ordering (variable ordering), one could give first preference to articulation bids, second preference to bids that articulate on the winning branch only, and third preference to bids that do not articulate on either branch (among them, price-based bid ordering could be used). One could also try to identify *sets* of bids that articulate the bid graph, and branch on all of the bids in the set. However, to keep the computation linear time in the size of G, CABOB simply gives first priority to articulation bids, and if there are none, uses other bid ordering schemes, discussed later. If there are several articulation bids, CABOB

branches on the one that is found first (the others will be found at subsequent levels of the search). One could also use a more elaborate scheme for choosing among articulation bids.

The COMPLETE special case

In step (1), CABOB checks whether the bid graph G is complete: $|E| = |\frac{n(n-1)}{2}|$. If so, only one of the remaining bids can be accepted. CABOB thus picks the bid with highest price, updates $\tilde{f}^*$ if appropriate, and prunes the search path.

The NO_EDGES special case

In step (1), CABOB checks whether the bid graph G has any edges ($|E| > 0$). If not, it accepts all of the remaining bids, updates $\tilde{f}^*$ if appropriate, and prunes the search path.

Preprocessing

Several preprocessing techniques have been proposed for search-based winner determination algorithms [Sandholm, 1999], and any of them could be used in conjunction with CABOB. However, in CABOB the search itself is fast, so we did not want to spend significant time preprocessing (since that could dwarf the search time). The only preprocessing that CABOB does is that as a bid B_x arrives, CABOB discards every bid B_y that B_x dominates ($p_x \geq p_y$ and $S_x \subseteq S_y$), and discards bid B_x if it is dominated by any earlier bid.

4 Bid Ordering Heuristics

In step (10) of CABOB, there are potentially a large number of bids on which CABOB could branch on. We developed several *bid ordering heuristics* for making this choice.[4]

- *Normalized Bid Price (NBP):* [Sandholm and Suri, 2000]. Branch on a bid with the highest $w_j = \frac{p_j}{(|S_j|)^\alpha}$. It was conjectured [Sandholm and Suri, 2000] that α slightly less than 0.5 would be best (since $\alpha = 0.5$ gives the best worst-case bound within a greedy algorithm [Lehmann *et al.*, 1999]), but we determined experimentally that $\alpha \in [0.8,\ 1]$ yields fastest performance.
- *Normalized Shadow Surplus (NSS):* The problem with NBP is that it treats each item as equally valuable. It could be modified to weight different items differently based on static prices that, e.g., the seller guesses before the auction. We propose a more sophisticated method where the items are weighted by their "values" *in the remaining subproblem.* We use the shadow price y_j from the remaining DUAL problem as a proxy for the worth of an item. We then branch on the bid whose price gives the highest surplus above the worth of the items (normalized by the worths so the surplus has to be greater if the bid uses valuable items): $w_j = \frac{p_j - \sum_{i \in S_j} y_i}{(\sum_{i \in S_j} y_i)^\alpha}$. Next we showed experimentally that the following modification to the normalization leads to faster performance: $w_j = \frac{p_j - \sum_{i \in S_j} y_i}{\log(\sum_{i \in S_j} y_i)}$. We call this scheme NSS.
- *Bid Graph Neighbors (BGN):* Branch on a bid with the largest number of neighbors in the bid graph G. The motivation is that this will allow CABOB to exclude the largest number of still eligible bids from consideration.
- *Number of Items (NI):* Branch on a bid with the largest number of items. The motivation is the same as in BGN.
- *One Bids (OB):* Branch on a bid whose x_j-value from LP is closest to 1. The idea is that the more of the bid is accepted in the LP, the more likely it is to be competitive.
- *Fractional Bids (FB):* Branch on a bid with x_j closest to 0.5. This strategy is advocated in the operations research literature [Wolsey, 1998]. The idea is that the LP is least sure about these bids, so it makes sense to resolve that uncertainty rather than to invest branching on bids about which the LP is "more certain". More often than not, the bids whose x_j values are close to 0 or 1 tend to get closer to those extreme values as search proceeds down a path, and in the end, LP will give an integer solution. Therefore those bids never end up being branched on.

4.1 Choosing Bid Ordering Heuristics Dynamically

We ran experiments on several distributions (discussed later) using each one of the heuristics as the primary heuristic, while using each of the other heuristics as a tie-breaker. We also tried using a third heuristic to break remaining ties, but that never helped. The best composite heuristic (OB+NSS) used OB first, and broke ties using NSS.

We noticed that on certain distributions, OB+NSS was best while on distributions where the bids included a large number of items, NSS alone was best. The selective superiority of the heuristics led us to the idea of choosing the bid ordering heuristic *dynamically based on the characteristics of the remaining subproblem.* We determined that the distinguishing characteristic between the distributions was LP density:

$$\text{density} = \frac{\text{number of nonzero coefficients in LP}}{\text{number of LP rows} \times \text{number of LP columns}}$$

OB+NSS was best when density was less than 0.25 and NSS was best otherwise. Intuitively, when the LP table is sparse, LP is good at "guessing" which bids to accept. When the table is dense, the LP makes poor guesses (most bids are accepted to a small extent). In those cases the price-based scheme NSS (that still uses the shadow prices from the LP) was better.

So, at every search node in CABOB, the density is computed, and the bid ordering scheme is chosen dynamically (OB+NSS if density is less than 0.25, NSS otherwise).

As a fundamentally different bid ordering methodology, we observe that stochastic local search—or any other approximate algorithm for the problem—could be used to come up with a good solution fast, and then that solution could be forced to be the left branch (IN-branch) of CABOB (with the "most sure" bids nearest the root) so as to give CABOB a more global form of guidance in bid ordering.

[4] This corresponds to variable ordering. Choosing between the IN-branch ($x_j = 1$) and the OUT-branch ($x_j = 0$) first corresponds to value ordering. In the current version of CABOB, we always try the IN-branch first. The reason is that we try to include good bids early so as to find good solutions early. This enables more pruning through upper bounding. It also improves the anytime performance. CPLEX, on the other hand, uses value ordering as well in that it sometimes tries the OUT-branch first. In future research we plan to experiment with that option in CABOB as well.

5 Design Philosophy of CABOB vs. CPLEX

We benchmarked CABOB against a general-purpose integer programming package, CPLEX 7.0. It was recently shown [Andersson *et al.*, 2000] that CPLEX 6.5 is faster (or comparable) in determining winners in combinatorial auctions than are the first-generation special-purpose search algorithms [Sandholm, 1999; Fujishima *et al.*, 1999]. CPLEX 7.0 is about 1.6 times faster than CPLEX 6.5, so when we compare CABOB against CPLEX 7.0, to our knowledge, we are comparing it against the state-of-the-art.

There are some fundamental differences between CABOB and CPLEX that we want to explain to put the experiments in context. CPLEX uses best-bound search (A*) [Wolsey, 1998] which requires exponential space (it also has an option to force depth-first search, but that makes CPLEX somewhat slower), while CABOB uses depth-first branch-and-bound (DFBnB) which runs in linear space. Thus, on some hard problems, CPLEX ran out of virtual memory. In our experiments we only show cases where CPLEX was able to run in RAM. DFBnB puts CABOB at a disadvantage when it comes to reaching the optimal solution fast since it does not allow CABOB to explore the most promising leaves of the search tree first. At the same time, we believe that the memory issue make A* unusable for combinatorial auctions in practice. Like CABOB, CPLEX uses LP to obtain upper bounds.

CPLEX uses a "presolver" to manipulate the LP table algebraically [Wolsey, 1998] to reduce it before search. In the experiments, we ran CABOB without any presolving. Naturally, that could be added to CABOB as a preprocessing step.

Put together, everything else being equal, CPLEX should find an optimal solution and prove optimality faster than DFBnB, but one would expect the anytime behavior to be worse.

6 Experimental Setup

We tested CABOB and CPLEX on the common combinatorial auction benchmarks distributions: those of [Sandholm, 1999], and the CATS distributions [Leyton-Brown *et al.*, 2000]. In addition, we tested them on new distributions.

The distributions from [Sandholm, 1999] are:

- **Random:** For each bid, pick the number of items randomly from $1, 2, ..., m$. Randomly choose that many items without replacement. Pick a price from $[0, 1]$.
- **Weighted random:** As above, but pick the price between 0 and the number of items in the bid.
- **Uniform:** Draw the same number of randomly chosen items for each bid. Pick the prices from $[0, 1]$.
- **Decay:** Give the bid one random item. Then repeatedly add a new random item with probability α until an item is not added or the bid includes all m items. Pick the price between 0 and the number of items in the bid. In the tests we used $\alpha = 0.75$ since the graphs in [Sandholm, 1999] show that this setting leads to the hardest instances on average (at least for that algorithm).

We tested the algorithms on all of the CATS distributions: **paths**, **regions**, **matching**, **scheduling**, and **arbitrary**. For each one of them, we used the default parameters in the CATS instance generators, and varied the number of bids.

We also tested the algorithms on the following new benchmark distributions:

- **Bounded:** For each bid, draw the number of items randomly between a lower bound and an upper bound. Randomly include that many distinct items in the bid. Pick the price between 0 and the number of items in the bid.
- **Components:** A number of independent problems, each from the uniform distribution.

We generate bids so no two bids have the same set of items. The experiments were conducted on a 933 MHz Pentium III PC with 512MB RAM. Each point in each plot is the median run time for 100 instances. CABOB and CPLEX both use the default LP solver that comes with CPLEX (dual simplex). CABOB and CPLEX were tested on the same instances.

7 Experimental Results

On the random distribution (Fig 1 left), CABOB is faster than CPLEX and the difference grows with the number of bids. The preprocessor of CABOB eliminates a large number of bids. CABOB always resorted to search while CPLEX's presolve+LP solved the problem without search on 47% of the instances. On the weighted random distribution (Fig 1 right), the performance of CABOB and CPLEX is almost identical. However, they achieve this very differently. With 2,000 bids, CPLEX's presolve+LP solves the problem 95% of the time while CABOB resorts to search 88% of the time.

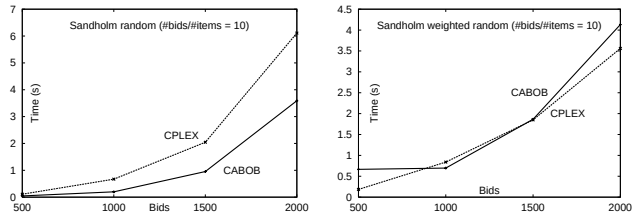


Figure 1: *Run times on the* **random** *and* **weighted random** *distributions.*

Interestingly, on the decay distribution (Fig. 2 left)—which is perhaps the most realistic distribution of those of [Sandholm, 1999], and was reported to be difficult for the earlier winner determination algorithms—both algorithms solve the problem using LP in almost all cases. CPLEX goes to search 2% of the time while CABOB resorts to search only 1% of the time. CABOB is faster than CPLEX, mainly because CPLEX uses presolve while CABOB does not.

On the uniform distribution (Fig. 2 right), both algorithms resort to search. The speeds are comparable, but CPLEX is faster. For the first-generation winner determination algorithms [Sandholm, 1999; Fujishima *et al.*, 1999], the instances with small numbers of items per bid were much harder than instances with long bids. For both CABOB and CPLEX, complexity is almost invariant to the number of items per bid, except that complexity *drops* significantly as the bids include less than 5 items each! This is because LP can handle cases with short bids well, both in terms of upper bounding and finding integer solutions. (If each bid contains only *one* item, LP *always* finds an integer solution).

The components distribution demonstrates the power of CABOB's decomposition technique and pruning across com-

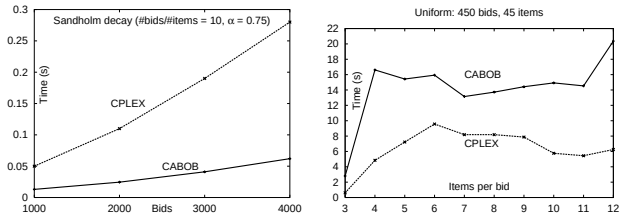


Figure 2: *Run times on the* **decay** *and* **uniform** *distributions.*

ponents. CABOB's run-time increases linearly with the number of components while CPLEX's time is exponential (Fig. 3). Already at 5 components, CPLEX ran out of virtual memory. The same performance would be observed even if there were a "glue" bid that included items from each component, since CABOB would identify that bid as an articulation bid.

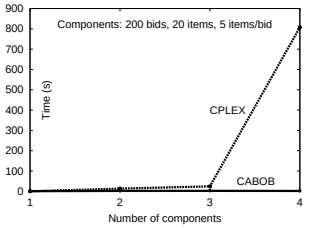


Figure 3: *Run times on the* **components** *distribution.*

On the bounded distribution (Fig. 4)—which is a more realistic version of the uniform distribution—the relative performance of CABOB and CPLEX depended on the bounds. For short bids, CPLEX was somewhat faster, but the *relative* speed difference decreased with the number of bids. For long bids, CABOB was much faster (mainly due to checking for completeness of the bid graph G), and the difference grew dramatically with the number of bids.

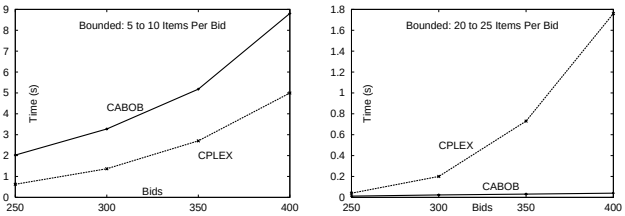


Figure 4: *Run times on the* **bounded** *distribution.*

Surprisingly, the CATS distributions were very easy. LP solved them in almost all cases. Interestingly, even the rare cases where the algorithms resorted to search disappeared as the number of bids *increased* (except for CATS arbitrary where the complexity did not vary much with the number of bids). As Fig. 5 shows, CABOB was faster than CPLEX (mainly because the preprocessor discards a large number of bids). The difference grows with the number of bids.

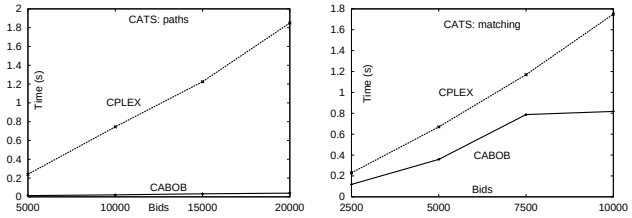


Figure 5: *Run times on CATS* **paths** *and* **matching**.

7.1 Anytime Performance

As expected from their designs, CABOB has better anytime performance than CPLEX. Fig. 6 shows a run that is typical in the sense of anytime performance, but which was carefully selected so that CABOB and CPLEX take equal time to prove that an optimal solution has been reached. CABOB dominates CPLEX throughout, and finds the optimal solution in 40% of the time it takes CPLEX to find it.

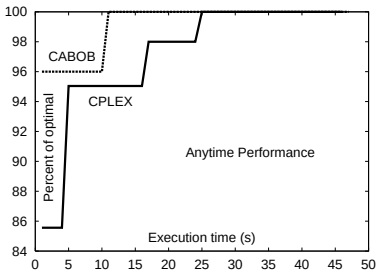


Figure 6: *Solution quality on the bounded distribution, reported by each algorithm once per second.*

8 Random Restarts

Random restarts have been widely used in local search algorithms, but recently they have been shown to speed up tree search algorithms as well [Gomes *et al.*, 1998]. We conjectured that random restarts, combined with randomized bid ordering, could avoid the perils of unlucky bid ordering. To see whether we could improve CABOB using random restarts, we implemented the random restarts methods that are best (to our knowledge) and improved them further to try to capitalize on the special properties of the problem.

We implemented the following restart strategies:

- *Double:* Double the execution time between restarts.
- *Constant:* Restart after every δ backtracks [Gomes *et al.*, 1998].
- *Luby-Sinclair-Zuckerman:* [Luby *et al.*, 1993] showed that the constant scheme above is optimal if δ is tailored to the run-time distribution, which is, unfortunately, usually not known in practice. Therefore, they constructed a scheme that suffers only an asymptotically logarithmic time penalty, independent of the run-time distribution. In the scheme, each run time is a power of 2. Each time a pair of runs of the same length has been executed, a run time of twice that length is immediately executed: $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, \ldots$.

We implemented the following bid ordering techniques to use with the restart strategies:

- *Random:* Randomly pick a remaining bid.
- *Boltzmann:* Pick a bid with probability $p_i = \frac{e^{\frac{q_i}{T}}}{\sum_j e^{\frac{q_j}{T}}}$, where $q_i = x_i + \frac{w_j}{LPUB}$. The value w_j is from the NSS bid ordering heuristic, and $LPUB$ is the objective function value from LP. Higher values of T result in more randomness.
- *Bound:* Each bid whose x_j value is within a bound b of the highest x_j value is equally probable.

We tried every bid ordering with every restart strategy, and varied the initial time allotment and the parameters δ, T, and b. CABOB was always faster than CABOB with restarts!

It turns out that this is not just a facet of our restart schemes or parameters settings. Random restarts tend to lead to speedup when the run-time distribution has a heavy tail [Gomes *et al.*, 1998]. We decided to test whether CABOB exhibits heavy-tailed run-times on the winner determination problem. We chose the distribution on which CABOB's run-time varied the most so as to increase the chance of finding a heavy tail. This was the uniform distribution with 5 items per bid. If a distribution has a heavy tail, the variance and usually also the mean are unbounded [Gomes *et al.*, 1998]. As can be seen in Fig. 7, our mean and variance are not only bounded, but constant. This means that the run-time distribution does not have a heavy tail. This explains our negative results with restarts, and suggests that random restarts are not a fruitful avenue for future improvement in this setting.

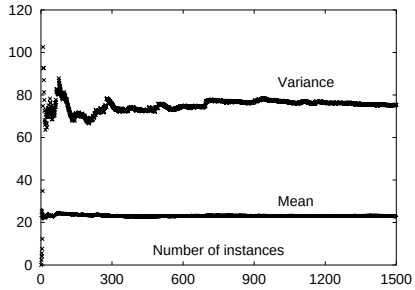


Figure 7: *The mean and variance of CABOB's search time as a function of the number of instances in the sample.*

9 Conclusions and Future Research

Combinatorial auctions where bidders can bid on bundles of items can lead to more economical allocations, but determining the winners is $\mathcal{NP}$-complete and inapproximable. We presented CABOB, a sophisticated search algorithm for the problem. It uses decomposition techniques, upper and lower bounding (also across components), a host of structural observations, elaborate and dynamically chosen bid ordering heuristics, and other techniques to increase speed—especially on problems with different types of special structure, which we expect to be common in real combinatorial auctions. Experiments against the fastest prior algorithm, CPLEX 7.0, show that CABOB is usually faster, never drastically slower, and in many cases drastically faster. Overall, this makes CABOB, to our knowledge, the currently fastest optimal algorithm for the problem. CABOB's search runs in linear space while CPLEX takes exponential space. CABOB also has significantly better anytime behavior than CPLEX.

We also uncovered interesting aspects of the problem itself. First, the problems with short bids that were hard for the first-generation of specialized algorithms are easy. Second, almost all of the CATS distributions are easy, and become easier with more bids. Third, we tested a number of random restart strategies, and showed that random restarts do not help on this problem because the run-time distribution does not have a heavy tail (at least not for CABOB).

We are currently working not only on designing faster algorithms for winner determination in combinatorial auctions, but also on winner determination in combinatorial reverse auctions and exchanges [Sandholm *et al.*, 2001], as well as in combinatorial markets with additional side constraints [Sandholm and Suri, 2001]. We are also developing methods for intelligent, selective elicitation of combinatorial bids from the market participants [Conen and Sandholm, 2001].

References

[Andersson *et al.*, 2000] Arne Andersson, Mattias Tenhunen, and Fredrik Ygge. Integer programming for combinatorial auction winner determination. In *ICMAS*, pages 39–46.

[Conen and Sandholm, 2001] Wolfram Conen and Tuomas Sandholm. Minimal preference elicitation in combinatorial auctions. In *IJCAI-2001 Workshop on Economic Agents, Models, and Mechanisms*.

[de Vries and Vohra, 2000] Sven de Vries and Rakesh Vohra. Combinatorial auctions: A survey. August 28th.

[Fujishima *et al.*, 1999] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJCAI*, pages 548–553.

[Gomes *et al.*, 1998] Carla Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *AAAI*.

[Hoos and Boutilier, 2000] Holger Hoos and Craig Boutilier. Solving combinatorial auctions using stochastic local search. In *AAAI*, pages 22–29.

[Lehmann *et al.*, 1999] Daniel Lehmann, Lidian Ita O'Callaghan, and Yoav Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. In *ACM Conference on Electronic Commerce* , pages 96–102.

[Leyton-Brown *et al.*, 2000] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce* , pages 66–76.

[Luby *et al.*, 1993] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47:173–180.

[Nisan, 2000] Noam Nisan. Bidding and allocation in combinatorial auctions. In *ACM Conference on Electronic Commerce* , pages 1–12.

[Rassenti *et al.*, 1982] S J Rassenti, V L Smith, and R L Bulfin. A combinatorial auction mechanism for airport time slot allocation. *Bell J. of Economics*, 13:402–417.

[Rothkopf *et al.*, 1998] Michael H Rothkopf, Aleksandar Pekeč, and Ronald M Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147.

[Sandholm and Suri, 2000] Tuomas Sandholm and Subhash Suri. Improved algorithms for optimal winner determination in combinatorial auctions and generalizations. In *AAAI*, pages 90–97.

[Sandholm and Suri, 2001] Tuomas Sandholm and Subhash Suri. Side constraints and non-price attributes in combinatorial markets. In *IJCAI-2001 Workshop on Distributed Constraint Reasoning*.

[Sandholm *et al.*, 2001] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. Winner determination in combinatorial auction generalizations. In *AGENTS Workshop on Agent-Based Approaches to B2B*.

[Sandholm, 1993] Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *AAAI*, p. 256–262.

[Sandholm, 1999] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *IJCAI*, pages 542–547, 1999. First appeared as Washington Univ., Dept. of Computer Science, WUCS-99-01, Jan. 28th.

[Sandholm, 2000] Tuomas Sandholm. eMediator: A next generation electronic commerce server. In *AGENTS*, pages 73–96, 2000. Early version: AAAI-99 Workshop on AI in Electronic Commerce, Orlando, FL, pp. 46–55, July 1999, and Washington University, St. Louis, Dept. of Computer Science WU-CS-99-02, Jan. 1999.

[Wolsey, 1998] Laurence Wolsey. *Integer Programming*. John Wiley.

A software architecture for dynamically generated adaptive Web stores*

Liliana Ardissono, Anna Goy, Giovanna Petrone and Marino Segnan
Dipartimento di Informatica - Università di Torino
Corso Svizzera 185, Torino, Italy
{liliana,goy,giovanna,marino@di.unito.it}

Abstract

We provide technical details about the software and hardware architecture of SETA, a prototype toolkit for the creation of Web stores which personalize the interaction with customers. SETA is based on a multi-agent architecture and on the use of MAS technologies, which support a seamless communication among the system agents and an easy distribution of such agents on different computers.

1 Introduction

Personalization has recently become a central focus of attention for Web-based systems [Riecken, 2000]. This paper describes the software and hardware architecture of SETA [Ardissono *et al.*, 1999; 2000], a prototype toolkit for the creation of adaptive Web stores, which tailor the suggestion and the presentation of products to the individual customer's needs. We will provide details about the SETA multi-agent architecture, the class hierarchy which defines the internal architecture of the system agents and the technologies used to support agent distribution, communication and specific agent activities. The main prototype store created using our system presents telecommunication products, like phones and faxes.

Sections 2 and 3 sketch the application domain and the adaptivity functionalities offered by the Web store. Section 4 describes multi-agent architecture. Sections 5 and 6 describe the management of multi-user access and communication among system agents. Sections 7 and 8 specify the internal design of a SETA agent and the external software used to obtain specific system functionalities. Section 9 provides technical details and section 10 compares our approach to the one of other commercial Web-based systems. Section 11 closes the paper.

*This paper describes the evolution of a system developed at the CS Department of the University of Torino, within the project "Servizi Telematici Adattativi", funded by Telecom Italia. We thank L. Console, L. Lesmo, C. Simone and P. Torasso for their comments to this work.

2 Application domain

We designed our system to satisfy personalization requirements in the Business to Customer area of e-commerce, focusing on the presentation of massively sold, medium-complexity products, such as home appliances. In this area, personalizing the presentation of goods is important because the customer has to select items out of a rich pool of alternatives. This decision depends on factors ranging from the price of the items to their features and the customer needs a lot of information to choose the item suiting her needs at best. Moreover, depending on her interests, not all the product features might be equally important: thus, the presentation can be dramatically improved by focusing on the most relevant information. Another reason for personalization is that the customer might not be aware of all the functionalities offered by a product class, therefore needing to be assisted in the identification of the relevant goods. Thus, the system should both personalize the presentation of the catalog and elicit information about the user's needs, in order to actively suggest alternative products.

Although e-commerce has strong adaptivity demands, it constraints the development of user interfaces in several ways. For instance, Web stores must be accessible via standard equipments, such as a personal computer, a (usually slow) Internet connection and a Web browser. Moreover, the time needed to browse the catalog and find the needed products must be as short as possible and the run-time efficiency of the system is essential. Finally, Web store shells, such as SETA, must satisfy the requirements of the store designer, possibly reducing the overhead in the configuration of a new Web store.

3 Adaptivity in SETA

A Web store catalog generated by SETA is organized as a hypertext containing two main page types:

(1) Pages presenting product classes in a synthetic style, specifying the main functionalities offered by the related items: e.g., the description of the faxes product class specifies that faxes transmit documents and some of them also make photocopies. These pages also provide the user with the hypertextual links for navigating the catalog.
(2) Pages showing detailed information about the items of a product class. These pages present the features offered by the individual items and provide the user with rich interaction functionalities: e.g., they enable her to ask for technical details, to create comparison tables "on the fly" and to put items into the shopping cart.

SETA dynamically generates the pages of a Web store catalog by exploiting personalization strategies for selecting the information to be presented on the basis of the user's interests and familiarity with the products. Moreover, the system presents the available items for a product class (e.g., the available fax models) sorting them on the basis of the user's preferences [Ardissono and Goy, 2000]. During the interaction, the system monitors the user's selections to identify her needs for product functionalities and suggest potentially interesting product classes which the user has not visited. In this way, the assistance is extended to the search for alternative products.

4 Architecture of SETA

The management of personalized interactions results from the coordination of several activities like user modeling, dynamic page generation, etc., requiring the exploitation of different knowledge and techniques; e.g., specific strategies are needed for tailoring layout, content and structure of the pages to the users. To address such complexity, modular architectures, integrating components devoted to the different tasks, are applied in the design of adaptive systems on the Web. The exploitation of agent-based technologies is even more effective, as MAS technologies support the development of distributed systems where heterogeneous agents offer specialized services and interact to produce the overall service in an open environment; e.g., [Jennings *et al.*, 1998; Giampapa *et al.*, 2000; Macho *et al.*, 2000].

In the development of our system, we have exploited knowledge representation techniques and agent-based technologies to improve the configurability of the toolkit and its scalability. SETA is based on a multi-agent architecture where specialized agents fill the main roles for the management of personalized interactions with customers; e.g., user modeling and generation of Web pages [Ardissono *et al.*, 1999]. Each agent handles multiple user sessions and maintains session-specific data in parallel session environments. In the following we sketch the most relevant agents.
At the Web store starting time, the **Session Manager** creates the SETA agents and provides them with the references to the other SETA agents they might need to send messages to. During the life-span of the Web store, the Session Manager handles the communication with the browsers, catching the user's actions and forwarding them to the Dialog Manager.

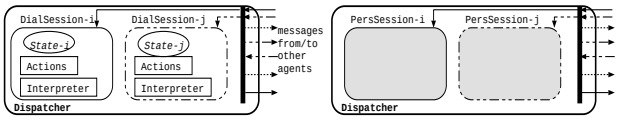


Figure 1: Parallel sessions within two SETA agents.

The **Dialog Manager** monitors the user's actions and maintains an interaction context storing information about the navigation in the catalog; after each user action, this agent choses the page to be produced next, asks the Personalization Agent to generate it and forwards the page to the Session Manager, to send it to the browser.
The **User Modeling Component (UMC)** maintains the models of customers, revising them during the interaction, on the basis of their behavior.
The **Product Extractor** supports the personalized suggestion of goods by ranking items on the basis of how strictly they match the user's preferences.
The **Personalization Agent** dynamically generates the code for the catalog pages by exploiting the information about the user provided by the UMC and a set of personalization rules for selecting information about products and type of description to be produced [Ardissono and Goy, 2000].

5 Management of parallel user sessions

The multi-user access to the Web store is managed by performing the session tracking within the Session Manager (a Servlet), and forwarding the session-specific messages to the agents of the architecture, so that they can process such messages and perform the related activities.

Each SETA agent is composed of an interface, the "Dispatcher", which handles the delivery and reception of messages, and of a set of *user-session agents* which maintain the session-dependent environments and carry on the activities to be performed within each user session; e.g., in Figure 1, the Dialog Manager and the Personalization Agent use two user-session agents each: "DialSession-i,j" and "PersSession-i,j". Each dispatcher acts as a wrapper and supports a uniform communication with the other SETA agents, by separating the communication flows related to the active sessions and forwarding incoming messages to the appropriate user-session agent. Moreover, the dispatcher creates and removes user-session agents, depending on the users connected to the store. The presence of dispatchers and user-session agents supports a simple management of session-dependent activities: user-session agents have separate

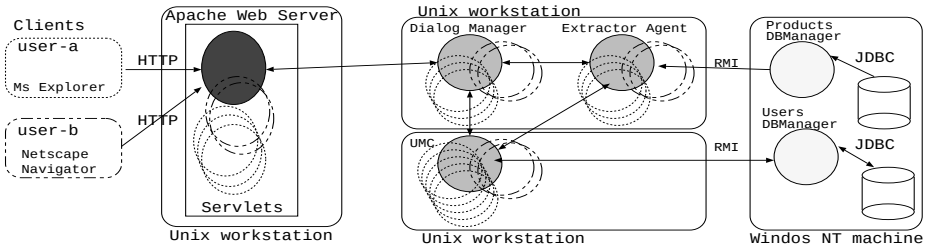


Figure 2: Parallel threads in the management of multiple user sessions.

internal states and work in parallel threads of execution within a SETA agent, performing services and activities related to the active sessions in an independent way.

6 Communication among SETA agents

As the number of distinct roles in the system architecture is fixed and well-determined, each agent knows which agents have to be contacted for requesting services and only needs their references; thus, there is no need to exploit middle agents: the Session Manager communicates such references after the creation of agents.

Our approach is integrated in the environment of the ObjectSpace Voyager tool [ObjectSpace, 2000], which we used to wrap the SETA agents, enabling them to run in parallel and to communicate via synchronous and asynchronous messages. The messages can be mapped to a subset of the KQML performatives. At the moment, the SETA agents do not need to communicate externally, therefore we did not comply with the FIPA ACL. According to the Voyager specification, the SETA agents exchange messages whose parameters contain, respectively, the name of the method to invoke and the array of its arguments. Voyager transforms the parameters of a message in a Java method call, so that the invocation of methods offered by a SETA agent is straightforward.[1]

Each Dispatcher is a Voyager Object and handles the incoming messages in parallel threads of execution, invoking the appropriate user-session agent to perform the requested task. As more than one message related to the same user session can be received at the same time, a user-session agent can perform parallel tasks; this fact raises mutual exclusion issues, resolved by inhibiting concurrent accesses to shared resources by means of Java synchronization facilities.

Figure 2 shows an example where two browsers ("user-a", represented as a dotted rounded square, and "user-b", a dashed one) access the system. The black oval represents the Servlet running within the HTTP server and

[1]For instance, an asynchronous message without return value, such as the following one: OneWay.invoke(agtReceiver, methodName, methodArgs); is translated by Voyager into: agtReceiver.methodName(methodArgs[0],methodArgs[1]).

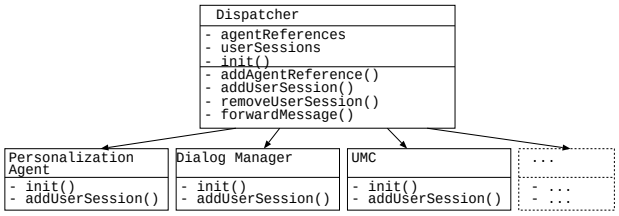


Figure 3: Class hierarchy defining a SETA agent.

the grey circles represent SETA agents: we have shown only the Dialog Manager, Extractor Agent and UMC. The figure shows multiple threads, related with the two active user sessions ("user-a", "user-b"), running within the HTTP Server and the SETA agents; the lines used to depict the threads correspond to the related sessions.

7 Design of a SETA agent

Figure 3 shows the class hierarchy defining the agents: the "Dispatcher" class specifies that a SETA agent has a list of references to the SETA agents it may send messages to ("agentReferences"); moreover, it has a set of user-session agents ("userSessions"). The class also offers the methods for initializing a SETA agent ("init()"), setting references of other SETA agents ("addAgentReference()"), creating and removing a user-session agent ("add/removeUserSession()") and forwarding messages to a user-session agent, to process a session-dependent request ("forwardMessage()"). The Dispatcher uses the communication facilities offered by Voyager for interacting with the other SETA agents.

Each SETA agent extends the "Dispatcher" class and may override the definition of its variables and methods. This is very useful for the definition of the user-session agents: to support the development of heterogeneous agents, specialized in the execution of different tasks, the SETA agents have to exploit user-session agents based on different technologies and designed following different approaches. For instance:

- In addition to the provision of services to the other SETA agents, the activities carried on by the Dia-

log Manager and the UMC include, respectively, the conditional execution of state transitions representing the evolution of the interaction with the user and the autonomous management of internal tasks. To satisfy these requirements, we have designed action-based user-session agents, where the agent state and the tasks to be performed are explicitly represented. Tasks are described as actions with preconditions determining the state conditions where they can be performed. In the Dialog Manager, the declarative representation of tasks supports an easy definition of the admissible state transitions in the interaction with the user. In the UMC, the presence of an interpreter selecting and performing actions, given the agent state, enables the agent to autonomously trigger its own internal activities, as soon as they can be performed [Ardissono *et al.*, 2000].

- In contrast, the action-based formalism is not suitable to other SETA agents, such as the Personalization Agent. In fact, the offered services can be handled by user-session agents responding to method invocations associated to potentially complex, but deterministic, tasks. Moreover, as such services must be based on the most recent information about the interaction with the user (situation of the user model, etc.), these agents do not need to explicitly manage an internal state: in fact, they have to retrieve such information from the other SETA agents, at each incoming request.

Figure 4 shows the class hierarchy of the action-based user-session agents defined in SETA. The "ActUserSessionAgent" class specifies their variables and generic behavior: it defines an explicit "state", representing the session-dependent data, an action list ("actionList") specifying the list of actions they can perform, and a pending list ("pendingList") used to store the suspended tasks. Moreover, the class offers an initialization method ("init()") and the "interpreter()". The user-session agents extend this class by redefining their own state (which may contain specific session-dependent data) and the action list, where their actions are listed. For clarity, the figure also shows the "Action" class, which provides the generic definition of an action specifying the basic components, i.e., the action type, preconditions, body and the method to check the preconditions, when the action is selected for execution; see [Ardissono *et al.*, 2000] for details.

8 Integration of heterogeneous software

We have integrated in the SETA agents external software tools for the management of very specific tasks. The modularity of the architecture enabled us to make them harmonically cooperate with the other components of the SETA agents. For instance, the UMC exploits the JESS rule-based system [Sandia, b] to maintain a structured

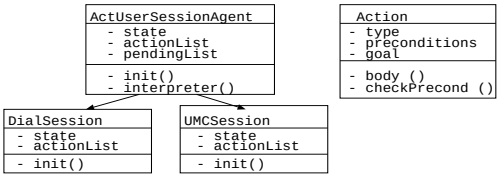


Figure 4: Class hierarchy defining an action-based user-session agent.

representation of the interaction context and to produce a synthetic description of the user's behavior: such description is then used within a Bayesian Network to reason about the user's interests and preferences.

The Natural Language Generator module of the Personalization Agent exploits JTg2 [DFKI and CELI, 2000] to generate the linguistic descriptions to be included in the Web pages presenting products and items. The JTg2 engine performs efficient and flexible generation in natural language. In its simplest conception it takes an object as input and returns a string. The crucial source of information is represented by the grammar rules: to integrate the engine in a new application, rules mapping the input object to the string have to be defined. The rules are augmented "if-then" statements spanning the input object top-down, left to right; to perform their task, the rules can access external modules that can be plugged into JTg2 to increase its own flexibility. In SETA, we have designed and implemented a set of NL grammar rules and two external modules: a test performer, which enables the engine to perform complex tests on the input object or on external sources, and a parameter getter, which enables the insertion of strings based on data coming from external sources. The input is a complex Java object representing a product and the output is the NL description to be included in the Web page. The grammar rules are used to fill in the slots in the templates defining descriptions.
JTg2 offers applicability tests to specify when a rule can be applied. In SETA, such tests are handled by the test performer and enable the grammar rules to be partially context-sensitive. In fact, they are essential to provide the NL Generator with the personalized content of the descriptions. Such content is represented by a selection of features, provided as a sorted list by the Personalization Agent on the basis of the contents of the user model. The applicability tests are used in SETA to integrate content personalization and NLG. In fact they make the NL Generator sensitive to the variation of input data (represented by the personalized list of product features to be described) and directly to the data in the user model; the NL Generator module, generates different linguistic descriptions, depending on the user's domain expertise. The use of NLG techniques reduces the amount of precompiled information to be defined at configuration time,

as the information about products is stored in a unique internal format. Moreover, it supports the generation of multilingual text (Italian and English) and the production of descriptions tailored to different user characteristics.

9 Technical details

The SETA system is implemented in Java (JDK 1.2) and is based on a Three-Tier architecture. The first tier runs on standard Java-enabled browsers, such as Netscape Navigator and Microsoft Explorer. In the second tier, the communication with the Web is supported by the Apache HTTP Server; moreover, we have exploited the functionalities offered by the Servlets to track the interaction with browsers. The bulk of the system resides in this tier and runs on a Unix environment: the SETA agents are distributed on two workstations for experimental purposes. The third tier includes the databases storing data about products and users and resides on a Windows NT computer. As shown in Figure 2, the communication between the second and the third tier is based on RMI (for historical reasons), while the agents in the middle tier communicate by exchanging Voyager messages.

10 Comments

10.1 Other three-tier architectures

As *n-tier* architectures are a popular solution for complex Web-based systems, the relation between our architecture and the other approaches has to be discussed, with specific reference to frameworks like J2EE [Sandia, a], which are industry standards for developing Web-based systems. The most important differences are in the organization of the middle tier of such architectures:

- The middle tier of SETA exploits Servlets for the communication with the Web, while it relies on the facilities offered by Voyager to exploit permanent distributed objects (SETA agents), which cooperate to carry on the interaction with the user. The whole logic for the generation of the Web pages resides in such permanent objects, with a specific object, the Dialog Manager, in charge of the selection of each interaction step, i.e., of choosing the next page to be produced, on the basis of the whole interaction context and of the last action performed by the user.
- The middle-tier of the J2EE framework uses Java Server Pages (JSPs) for the communication with the Web, and it uses Enterprise Java Beans (EJBs) to implement the other modules of the system.

The combined JSP-EJB paradigm is suited to a page-centric Web based system, where the interaction with the user is mostly based on the presentation of forms to be filled in and of pages containing the results of a (possibly complex) query. The typical content of such pages is the presentation of the results of a query to a database.

In contrast, the interaction with a Web store developed using SETA is not limited to a question-answer sequence: the user can navigate the catalog handling parallel contexts, searching for goods addressed to several beneficiaries during the same session. The system maintains parallel navigation contexts and supports the user in the switch among them. Moreover, the system can take the initiative and suggest alternative products to be analyzed, possibly interrupting the user's navigation. All these functionalities require the presence of an agent specialized in the management of the dialog with the user: this agent has to identify the next interaction step by using a declarative representation of the admissible turn sequences (e.g., a Finite State Automaton).

Of course, within a commercial application development, JSPs have a noticeable advantage with respect to pure Servlet-based approaches: the exploitation of JSPs facilitates the cooperation between Web authors and Java developers, letting Web authors concentrate on the development of HTML templates within the JSPs and the Java developers write the EJBs implementing DB access, legacy system interfaces, and so forth.

10.2 Technical user-interface issues

Although the exploitation of Applets has enhanced the functionalities offered by the user interface of SETA, we have experienced serious drawbacks. The most relevant problem is the fact that, to run an Applet containing non basic User Interface components (SWING), the browsers need to use plug-ins, sometimes incompatible with those installed on the computers. This requirement could make the access to the Web store complex and time-consuming, seriously reducing the portability of the system.

A second drawback concerns the combined use of Servlets and Applets within a system. On the one hand, the Servlets used to send HTML code to browsers can only receive String return values; on the other hand, for security purposes, Applets downloaded outside a LAN can only return values to the same HTTP Server from which they are downloaded, and firewalls forbid the communication with RMI servers. Together, these constraints impose that, whenever complex objects have to be returned by an Applet as the results of the decentralized interaction with the user, different Servlets within the same HTTP Server have to be exploited in the same application: one Servlet will forward Web pages to the browser and the other one will catch the complex return values produced by Applets. This approach has a subtle impact on the capability to track the state of the user sessions because it introduces parallel interaction flows between the browsers and the HTTP Server.

11 Conclusions

We have provided architectural and technological insights of SETA, a prototype toolkit for the development

of adaptive Web stores developed at the CS Department of the University of Torino. While this architecture has been described at the abstract level in [Ardissono *et al.*, 1999; 2000], this paper specifies the implementation of the system and the class hierarchy underlying the definition of the agents composing the multi-agent architecture.

In the development of our system, we exploited a basic and light agent-building tool, such as Voyager, to manage a seamless communication among agents, but we preferred to design our own infrastructure for developing the system agents because, as SETA is a specialized architecture for the creation of Web stores, it does not need the full capabilities offered by general-purpose agent-building tools, which typically provide facilities for agent communication, coordination, self-diagnosis, mobility, and many other functionalities. For example, SETA does not support the development of open systems interacting with middle agents. Thus, popular coordination models, such as ABS [Barbuceanu and Teigen, 1999], exceed the demands of our application example, because they are focused on a more complex issue, i.e., describing the external behavior of social agents. Other tools for the development of multi-agent systems, such as DECAF [Graham and Decker, 2000], seem to exceed our needs as well, as they support complex activities such as the coordination of a multiagent system to reach non-local goals and real time flexibility in the execution of tasks. In our system one agent is associated to each main role of the architecture; thus, we do not need to exploit schedulers for distributing tasks among alternative agents.

Scalability is a critical aspect and concerns several issues, among which load balancing. Being SETA a prototype, we did not explicitly address such aspect in our implementation; however, this problem can be bypassed by using the Voyager 3.3 Professional edition, which provides services that would take care of this issue.

We are now working to enhance the configurability of SETA, to support its instantiation on new domains for creating new Web stores, or generic recommender systems. A graphical tool currently enables the store designer to introduce the knowledge about customer classes (Stereotype KB) without writing any Java code. Moreover, the knowledge base containing information about products and their features, is automatically created by the system, given the structure of the Products DB (which contains a classification of items in product classes). Finally, we are integrating XML-based representations of the content of the Web pages generated by the system: different page types are defined in a DTD (Document Type Definition) and XSLT (eXtended Stylesheet Language Transformations) are used to produce the final user interface, on the basis of the personalized content of the page, encoded as an XML object: in the simplest case, such interface is generated as HTML code.

References

[Ardissono and Goy, 2000] L. Ardissono and A. Goy. Tailoring the interaction with users in web stores. *User Modeling and User-Adapted Interaction*, 10(4):251–303, 2000.

[Ardissono *et al.*, 1999] L. Ardissono, C. Barbero, A. Goy, and G. Petrone. An agent architecture for personalized web stores. In *Proc. 3rd Int. Conf. on Autonomous Agents*, pp. 182–189, Seattle, WA, 1999.

[Ardissono *et al.*, 2000] L. Ardissono, A. Goy, G. Petrone, and M. Segnan. Configurability within a multi-agent web store shell. In *Proc. 4th Int. Conf. on Autonomous Agents*, pp. 146–147, Barcelona, 2000.

[Barbuceanu and Teigen, 1999] M. Barbuceanu and R. Teigen. Higher level integration by multi-agent architectures. In P. Bernus, ed., *Handbook of Information System Architectures*. Springer Verlag, 1999.

[DFKI and CELI, 2000] DFKI and CELI. JTg2. www.celi.it/english/tecnologia/tecLing.html/, 2000.

[Giampapa *et al.*, 2000] J.A. Giampapa, M. Paolucci, and K. Sycara. Agent interoperation across multiagent system boundaries. In *Proc. of 4th Int. Conf. on Autonomous Agents*, pp. 179–186, Barcelona, 2000.

[Graham and Decker, 2000] J. Graham and K. Decker. Tools for developing and monitoring agents in distributed multi agent systems. In *Proc. of the Agents'2000 workshop on Infrastructure for scalable multi-agent systems*, Barcelona, 2000.

[Jennings *et al.*, 1998] N.R. Jennings, K.P. Sycara, and M. Wooldridge. A roadmap of agent research and development. In *Autonomous Agents and Multi-agent Systems*, pp. 275–306. Kluwer Academic Publishers, Boston, 1998.

[Sandia, a] Sandia National Laboratories. Java 2 Platform Enterprise Edition. http://java.sun.com/j2ee/.

[Sandia, b] Sandia National Laboratories. JESS, the Java Expert System Shell. http://herzberg.ca.sandia.gov/jess/.

[Macho *et al.*, 2000] S. Macho, M. Torrens, and B. Faltings. A multi-agent recommender system for planning meetings. In *Proc. of the Agents'2000 workshop on Agent-based recommender systems (WARS'2000)*, Barcelona, 2000.

[ObjectSpace, 2000] ObjectSpace. Voyager. http://www.objectspace.com/index.asp, 2000.

[Riecken, 2000] D. Riecken, editor. *Special Issue on Personalization*, volume 43. 2000.

Modularity and Design in Reactive Intelligence

Joanna J. Bryson and Lynn Andrea Stein
MIT Artificial Intelligence Laboratory, NE43-833
Cambridge, MA 02139, USA
joanna@ai.mit.edu, lynn.stein@olin.edu

Abstract

Software design is the hardest part of creating intelligent agents. Therefore agent architectures should be optimized as design tools. This paper presents an architectural synthesis between the three-layer architectures which dominate autonomous robotics and virtual reality, and a more agent-oriented approach to viewing behavior modules. We provide an approach, Behavior Oriented Design (BOD), for rapid, maintainable development. We demonstrate our approach by modeling primate learning.

1 Introduction

The last decade of research has shown impressive convergence on the gross characteristics of software architectures for complete agents such as autonomous robots or virtual reality (VR) characters [Kortenkamp *et al.*, 1998; Sengers, 1999; Bryson, 2000a]. The field is now dominated by 'hybrid', three-layer architectures [Gat, 1998]. The hybrids combine: (1) *behavior-based* AI, the decomposition of intelligence into simple, robust, reliable modules, (2) *reactive planning*, the ordering of expressed actions via carefully specified program structures, and (optionally) (3) *deliberative planning*, which may inform or create new plans or behaviors.

In this paper, we take the view that software design and methodology are critical to the advances that have been made in complete agents. We contribute architectural features that further facilitate the human engineering of these agents, including the effort to incorporate reliable learning and planning into the agent. We do this by enhancing the programmability of reactive plans and reintroducing modularity to the organization of the 'plan primitives'. In our system, plan primitives are not themselves modules, but *interfaces* to semi-autonomous behavior modules encapsulating specialized state for learning and control. This brings hybrid architectures closer to multi-agent systems (MAS) and behaviors closer to active objects [van Eijk *et al.*, 2001]. We also present a methodology for constructing complete agents in the architecture we describe, and some example agents.

2 Intelligence by Design

One of the most important aspects of the reactive revolution of the late 1980's is often overlooked. The break-throughs in robotics associated with reactive and behavior-based systems are usually attributed to the loss of deliberate planning and/or explicit representations. The real contribution of the reactive paradigm was explained nearly a decade earlier: you can't learn something you don't practically already know [Winston, 1975], nor, by extension, plan something you can't nearly already do. The reason is simple combinatorics [Chapman, 1987; Wolpert, 1996]. As evolutionary linguists and case-based reasoning researchers often tell us, what makes humans so intelligent are our exceptional ability to store and transmit solutions we manage to find [e.g. Hammond, 1990; Knight *et al.*, 2000].

Reactive and behavior-based AI thus facilitate the advance of AI in two ways. First, by severely deprecating both planning and state (and consequently learning), the reactive approach increased by default the emphasis on one of the largest problems of AI and software in general: design. Second, the behavior-based approach made fashionable a proven software design methodology: modularity.

The primary contributions of this paper are methodological. We provide more productive ways of creating hybrid systems. This is not to say that developing good software is ever easy, or that learning or productive planning should not be used to the fullest extent practical. In fact, we emphasize the role of learning in our methodology. What we *are* saying is that we favor approaches to hybrid systems that facilitate human design, because humans designers do most of the hard work in artificial intelligence.

3 Fundamentals of Reactive Plans

The terms 'reactive intelligence', 'reactive planning' and 'reactive plan' appear to be closely related, but actually signify the development of several different ideas. *Reactive intelligence* controls a reactive agent — one that can respond very quickly to changes in its situation. Reactive intelligence has sometimes been equated with statelessness, but that association is exaggerated. Reactive intelligence *is* associated with minimal representations and the lack of deliberation.

Reactive planning is something of an oxymoron. It describes the way reactive systems handle the problem traditionally addressed by conventional planning: action selection. Action selection is the ongoing problem (for an autonomous agent) of deciding what to do next. Conventional deliberate planning assumes the segmentation of intelligent behavior

into the achievement of discrete goals. A deliberate planner constructs a sequence of steps guaranteed to move an agent from its present state toward a goal state. Reactive planning, in contrast, chooses only the immediate next action, and bases this choice on the current context. In most architectures utilizing this technique, reactive planning is facilitated by the presence of *reactive plans*. Reactive plans are stored structures which, given the current context (both internal and environmental), specify the next act.

We will quickly address the concerns some researchers have with reactive planning. Hierarchical plans and centralized behavior arbitration *are* biologically plausible [Byrne and Russon, 1998; Prescott *et al.*, to appear]. They are sufficiently reactive to control robots in complex dynamic domains [e.g. Kortenkamp *et al.*, 1998] and have been shown experimentally to be as reactive as non-hierarchical, decentralized systems [Bryson, 2000b]. Although they do provide a single failure point, this can be addressed either by standard MAS techniques [e.g. Bansal *et al.*, 1998], or accepted as a critical system, like a power supply or a brain. Finally, as demonstrated by coordinated MAS and in Section 4 below, they do not preclude semi-autonomous behavior modules operating in parallel.

3.1 Basic Elements of Reactive Plans

Reactive plans support action selection. At any given time step, most agents have a number of actions which could potentially be expressed, at least some of which cannot be expressed simultaneously, for example sitting and walking. In architectures without centralized action selection, [e.g. Arkin, 1990; Maes, 1990], the designer must fully characterize *for each action* how to determine when it might be expressed. For engineers, it is generally easier to describe the desired behavior in terms of sequences of events. This strategy is complicated by the non-determinism of environments. Several types of events may interrupt the completion of an intended action sequence. These events fall into two categories: (1) some combination of alarms, requests or opportunities may make pursuing a different plan more relevant, and (2) some combination of opportunities or difficulties may require the current 'sequence' to be reordered. We have determined 3 element types for reactive plans which, when combined, support both of these situations.

Simple Sequences

The first element type is a simple sequence of primitive actions: $\iota_1, \iota_2, \ldots \iota_n$. In our own architecture, we call this element an *action pattern*. Including the sequence as an element type is useful for two reasons. First, it allows an agent designer to keep the system as simple as possible, which both makes it more likely to succeed, and communicates more clearly to a subsequent designer the expected behavior of that plan segment. Second, it allows for speed optimization of elements that are reliably run in order, which can be particularly useful in sequences of preconditions or in fine motor control.

Executing a sequential plan involves priming or activating the sequence, then releasing for execution the first primitive act ι_1. The completion of any ι_i releases the following ι_{i+1} until no active elements remain. Notice that this is *not* equivalent to the process of *chaining*, where each element is essentially an independent production, with a precondition set to the firing of the prior element. A sequence is an additional piece of control state; its elements may also occur in different orders in other sequences.

Basic Reactive Plans

The next element type supports the case when changes in circumstance can affect the order in which a plan is executed. We developed this idiom independently, and called it a *competence*. However, it occurs in a number of other architectures [e.g. Fikes *et al.*, 1972; Nilsson, 1994; Correia and Steiger-Garção, 1995] and is so characteristic of reactive planning, that we refer to the generic idiom as a *Basic Reactive Plan* or BRP.

A *BRP step* is a tuple $\langle \pi, \rho, \alpha \rangle$, where π is a priority, ρ is a releaser, and α is an action. A *BRP* is a small set (typically 3–7) of plan steps $\{\langle \pi_i, \rho_i, \alpha_i \rangle *\}$ associated with achieving a particular goal condition. The releaser ρ_i is a conjunction of boolean perceptual primitives which determine whether the step can execute. Each priority π_i is drawn from a total order. Each action α_i may be either another BRP or a sequence as described above.

The order in which plan steps are expressed is determined by two means: the releaser and the priority. If more than one step is operable, then the priority determines which step's α is executed. If no step can fire, then the BRP terminates. The top priority step of a BRP is often, though not necessarily a goal condition. In that case, its releaser, ρ_1, recognizes that the BRP has succeeded, and its action, α_1 terminates the BRP.

The details of the operation of a BRP are best explained through an example. BRPs have been used to control such complex systems as mobile robots and flight simulators [Correia and Steiger-Garção, 1995; Benson, 1996; Bryson and McGonigle, 1998]. However, for clarity we draw this example from blocks world. Assume that the world consists of stacks of colored blocks, and that an agent wants to hold a blue block.

$$\uparrow \left\langle \begin{array}{cr} \underline{\text{Priority}} & \underline{\text{Releaser}} \Rightarrow \underline{\text{Action}} \\ 4 & \text{(holding) (held 'blue)} \Rightarrow \textit{goal} \\ 3 & \text{(holding)} \Rightarrow \text{drop-held, lose-fix} \\ 2 & \text{(fixed-on 'blue)} \Rightarrow \text{grasp-top-of-stack} \\ 1 & \text{(blue-in-scene)} \Rightarrow \text{fixate-blue} \end{array} \right\rangle \quad (1)$$

In this case priority is strictly ordered and represented by position, with the highest priority step at the top. We refer to steps by priority.

This single reactive plan can generate a large number of expressed sequential plans. In the initial context of a red block stacked on a blue block, we might expect the plan 1–2–3–1–2–4 to execute. But if the agent is already fixated on blue and fails to grasp the red block successfully on first attempt, the expressed plan would look like 2–1–2–3–1–2–4. If the unsuccessful grasp knocked the red block off the blue, the expressed plan might be 2–1–2–4. This BRP is identically robust and opportunistic to changes caused by another agent.

If an action fails repeatedly, then the above construction might lead to an indefinite behavior cycle. This can be prevented through several means. Our competences allow a retry

limit to be set at the step level. Thus a *competence step* is really a quadruple $\langle \pi, \rho, \alpha, \eta \rangle$, where η is an optional maximum number of retries. Other systems often have generic rules for absence of progress or change.

The most significant feature of a BRP is that it is relatively easy to engineer. To build a BRP, the developer imagines a worst-case scenario for solving a particular goal. The priorities are then set in the inverse order that the steps might have to be executed. Next, preconditions are set, starting from the highest priority step, to determine whether it can fire. For each step, the preconditions are simplified by the assurance that the agent is already in the context of the current BRP, and that no higher priority step can fire.

Plan Manipulation

Finally, a reactive system must be able arbitrate *between* plans. We do this with a third element type called a *drive collection*, also based on the BRP. A 'step', or in this case, *drive element*, now has five elements $\langle \pi, \rho, \alpha, A, \nu \rangle$. For a drive, the priority and releaser π and ρ are as in a BRP, but the actions are different. *A* is the *root* of a BRP hierarchy, while α is the *currently active* element of the drive. If a drive element is selected for action, but its α is null because a BRP or sequence has just terminated, then α is set to the *A* for that drive. The drive element begins action selection again from the root of its hierarchy.

This system improves reaction time by eliminating the stack that might be produced when traversing a plan hierarchy. On every program cycle, the agent checks only the drive-collection priorities, and at most one other set of priorities, if α is currently a BRP rather than a sequence. It also allows the agent to periodically re-traverse its decision tree and notice any context change. This approach also allows the hierarchy of BRPs to contain cycles or oscillations, which are frequently useful patterns of behavior. Since there is no stack, there is no *obligation* for a competence chain to terminate.

The fifth member of a drive element, ν, is an optional maximum *frequency* at which this element is visited. This is a convenience for clarity, like the retry limit η on the competence steps — either could also be controlled through preconditions. The frequency in a real-time system sets a temporal limit on how frequently a drive element may be executed. For example, on a mobile robot [Bryson and McGonigle, 1998] we had the highest priority drive-element check robot's battery level, but this was only executed every two minutes. The next highest priority was checking the robot's sensors, which happened at 7Hz. Other, lower-priority processes then used the remaining interspersed cycles.

One further characteristic discriminates drive collections from competences / BRPs. Only one element of a competence is expected to be operating at any one time, but for a drive collection, multiple drives may be effectively active simultaneously. If a high-priority drive takes the attention of the action-selection mechanism, the program state of any active lower drive is preserved. In the case of our robot, if the navigation drive is in the process of selecting a destination when the battery needs to be checked, attention returns to the selection process exactly where it left off once the battery drive is finished. Further, action primitives in our system are not stand-alone, consumatory acts, but are interfaces to semi-autonomous behaviors which may be operating in parallel (see Section 4 below.) Thus the action 'move' in the robot's script merely confirms or transmits current target velocities to already active controllers. A moving robot does not stop rolling while its executive attends to its batteries or its sensors.

3.2 Discussion — Other Reactive Architectures

We refer to reactive plan structures as described above as Parallel-rooted, Ordered Slip-stack Hierarchical (POSH) action selection. Although we freely distribute implementations of this architecture in both C++ and Lisp / CLOS, we have also implemented versions of POSH action selection in other architectures [Bryson and Stein, 2001].

The functionality of the BRP, which in our experience is a critical element of reactive planning, is surprisingly missing from several popular architectures. In effect, architectures using middle layers like PRS [Georgeff and Lansky, 1987] seem to expect that most of behavior can be sequenced in advance, and that being reactive is only necessary for dealing with external interruptions by switching plans. On the other hand, architectures such as subsumption [Brooks, 1991] or ANA [Maes, 1990] expect that there is so *little* regularity in the arbitration of behavior that all actions must be considered for execution at all times. We have found the most expedient solution to the design problem of reactive planning is to categorize selection into things that need to be checked regularly, things that only need to be checked in a particular context, and things that one can get by not checking at all. These categories correspond to our three types of plan elements: drive collections, competences, and action patterns.

4 Semi-Autonomous Behavior Modules and the Role of Perceptual State

Besides emphasizing the use of modularity, the behavior-based movement also made an important engineering contribution by emphasizing specialized learning [e.g Brooks, 1991, pp. 158–9]. Specializing learning increases its probability of success, thus increasing its utility in a reliable agent. Similarly, modularity simplifies program design, at least locally, thus increasing the probability of correctness. Governing the interaction of multiple independent behavioral modules can be a difficulty, but we have already addressed this issue in the previous section.

Consider this description of standard hybrid systems:

> The three-layer architecture arises from the empirical observation that effective algorithms for controlling mobile robots tend to fall into three distinct categories: (1) reactive control algorithms which map sensors directly onto actuators with little or no internal state; (2) algorithms for governing routine sequences of activity which rely extensively on internal state but perform no search; and (3) time-consuming search-based algorithms such as planners. [Gat, 1998, p. 209]

Gat's view of three-layer architectures is particularly close to our own view of agent intelligence, because it puts con-

trol firmly in the middle, reactive-plan layer. The deliberative 'layer' operates when prompted by requests. However, we differ on the notion that there are many actions which can really map sensors to actuators with little internal state or consideration for the past.

Nearly all perception is ambiguous, and requires expectations rooted in experience to discriminate. For a mobile robot, this 'experience' may be from the last half a second, for discriminating sonar 'ghosts', half a minute, to move around a bumped object invisible to sonar, or days, as in remembering a local map. Primitive actions governed by the reactive plans may depend on any of this information. We do not believe this information should be passed between 'layers' either by micro-management or as parameters. Rather, in our model, the primitives of a reactive plan interface directly to semi-autonomous behavior modules. Each module maintains its own state and may possibly perform its own 'time-consuming' processes (such as memory consolidation or search) in parallel to the main activity of the complete agent. Thus our view of agent control is very similar to Gat's, except that (1) we increase the number, specificity and potential simplicity of the modules composing his top layer, and (2) we replace the notion of a bottom layer with the that of an interface between the action selection module of the robot and its (other) behavior modules.

5 Developing an Agent

The process of developing an agent with these two attributes, POSH action selection and a behavior library, we call Behavior Oriented Design. The analogy between BOD and OOD is not limited to the metaphor of the behavior and the object, nor to the use of methods on the behavior objects as primitives to the reactive plans. The most critical aspect of BOD is its emphasis on the design process itself. The old problem of behavior decomposition (and new, analogous one for MAS) is solved by using state requirements, as in modern object decomposition. Also as in OOD, BOD emphasizes cyclic design with rapid prototyping. The process of developing an agent alternates between developing libraries of behaviors, and developing reactive plans to control the expression of those behaviors.

5.1 The Initial Decomposition

The steps of initial decomposition are as follows. (1) Specify at a high level what the agent is intended to do. (2) Describe likely activities in terms of sequences of actions (prototype reactive plans.) (3) Identify sensory and action primitives from these sequences. (4) Identify the state necessary for these primitives, clustering them by shared state (prototype behaviors). (5) Identify and prioritize goals or drives that the agent may need to attend to (prototype POSH drive roots). (6) Select a first behavior to implement.

5.2 The Development Process

The remainder of the development process is not linear. It consists of the following elements, applied repeatedly as appropriate: coding behaviors, coding reactive plans, testing and debugging code, and revising the specifications made in the initial phase.

Usually only one behavior and one reactive plan will be actively developed at a time. We strongly suggest maintaining the lists developed in the initial phase as documentation. Where possible, such documentation should be part of active code. For example, the primitive list should be a file of code specifying the interface calls. Similarly, old reactive plans should be preserved with their development history and used as a test suite as modifications are made to the behavior libraries.

5.3 Revising the Specifications

The most interesting part of the BOD methodology is the set of rules for revising the specifications. The main design principle of BOD is *when in doubt, favor simplicity.* A primitive is preferred to an action sequence, a sequence to a competence. Heuristics then indicate when the simple element must be decomposed into a more complex one. One guiding principle is to reduce redundancy. If a particular plan or behavior can be reused, it should be. If only part of a plan or a primitive action can be used, then a change in decomposition is called for. In the case of an action primitive, the primitive should be decomposed into two or more primitives, and the original action replaced by a sequence. If a sequence sometimes needs to contain a cycle, or often does not need some of its elements to fire, then it should really be a competence. A new plan element should have the same name and functionality as the primitive or sequence it replaces. This allows established plans to continue operating without change.

Reactive plan elements should not require long or complex triggers. Perception should be handled at the behavior level; it should be a skill. A large number of triggers should be converted into a single perceptual primitive. Another problem that crops up in competence design can be the presence of too many elements. More than seven elements in a competence, or difficulty in appropriately prioritizing or setting triggers, indicates that a competence needs to be decomposed into two. If several of the elements can be seen as working to complete a subgoal, they may be moved to a child competence. If two or more of the elements always follow each other in sequence, they should be converted into an action pattern. If the competence is actually trying to achieve its goal by two different means, then it should be broken into two sibling competences which are both inserted into the original competence's parent.

6 Example: Modeling Transitivity in a Non-Human Primate

Although we have used our methodology on a mobile robot [Bryson and McGonigle, 1998] and on virtual reality characters [Bryson and Thórisson, 2001], we find artificial life (ALife) more conducive for quantitative comparisons [e.g. Bryson, 2000b] and for communicating with researchers. We thus illustrate BOD by building an ALife model of primate intelligence. Unfortunately, this shows only degenerate examples of drive collections, but space is limited.

We begin by modeling the ability of a monkey to perform transitive inference [McGonigle and Chalmers, 1977]. This

task is interesting, because it was initially believed to require reasoning, which was in turn considered to require language. Squirrel monkeys (*saimiri scuireus*) were trained on four pairs of colored pucks; AB, BC, CD, and DE; to favor the earlier element of the pair. Transitivity is the ability to generalize this pairing spontaneously (without further training) when first presented with the novel pairs such as AC, AD, BD and so on. The behavior of the monkeys on this task has already been modeled by Harris and McGonigle [1994], who modeled the transitive capability as a prioritized stack of production rules of the type 'avoid E' or 'select A'. Based on the nature of the errors the monkeys made, and their separate performance on three-item sets, different stacks were built representing the rules learned by each monkey.

We begin by replicating a simplified version of Harris's system modeling a skilled monkey. To simplify the simulation task, we model both the monkey and its testing environment as a single intelligent agent with two behaviors. The 'monkey' has two pieces of variable state — its hand and its visual attention, the test box has only a test-board for holding two or three items.

$$\textit{life} \Rightarrow \left\langle\left\langle \begin{array}{r}(\text{no-test}) \Rightarrow \text{new-test}\\ (\text{grasping}) \Rightarrow \text{finish-test}\\ \Rightarrow \textbf{elvis-choice}\end{array}\right\rangle\right\rangle$$

$$\textbf{elvis-choice} \Rightarrow \left\langle \begin{array}{r}(\text{see-red}) \Rightarrow \textbf{noisy-grasp}\\ (\text{see-white}) \Rightarrow \text{grasp-seen}\\ (\text{see-blue}) \Rightarrow \text{grasp-seen}\\ (\text{see-green}) \Rightarrow \text{grasp-seen}\end{array}\right\rangle \qquad (2)$$

$$\textbf{noisy-grasp} \Rightarrow \langle \text{screech} \rightarrow \text{grasp-seen}\rangle$$

This plan gives an example of each element type described in Section 3, though the action pattern is gratuitous. Priority is again listed from the top. The drive collection, life, has no goal, so it never ends. New-test is a call to the test behavior which randomly resets the test board; finish-test clears it. The seeing primitives all map to a single method on the monkey behavior, which performs a 'visual' scan of the test board and leaves visual attention on an appropriately colored object, if it exists. Grasp-seen, also a method on the monkey behavior, is thus able to use deictic reference without variable passing.

We now enhance the model by forcing the monkey to learn the ordering of the pucks. This also requires augmenting the test-behavior to reward the monkey appropriately.

$$\textit{life} \Rightarrow \left\langle\left\langle \begin{array}{r}(\text{no-test}) \Rightarrow \text{new-test}\\ (\text{rewarded}) \Rightarrow \textbf{end-of-test}\\ (\text{grasping}) \Rightarrow \textbf{elvis-reward}\\ \Rightarrow \textbf{educated-grasp}\end{array}\right\rangle\right\rangle$$

$$\textbf{elvis-reward} \Rightarrow \left\langle \begin{array}{r}(\text{find-red}) \Rightarrow \text{reward-found}\\ (\text{find-white}) \Rightarrow \text{reward-found}\\ (\text{find-blue}) \Rightarrow \text{reward-found}\\ (\text{find-green}) \Rightarrow \text{reward-found}\end{array}\right\rangle \qquad (3)$$

$$\textbf{educated-grasp} \Rightarrow \langle \text{adaptive-choice} \rightarrow \text{grasp-seen}\rangle$$

$$\textbf{end-of-test} \Rightarrow \langle \text{consider-reward} \rightarrow \text{finish-test}\rangle$$

We add a new behavior, sequence learning, to the system, which though ascribable to the monkey, is separate from the existing monkey-body behavior. The sequence-learner contains a list of known objects with weights, a 'significant difference' and a 'weight shift'. If the monkey is correct in its discrimination, but its certainty was less than significant-difference, then consider-reward adds weight-shift to the weight of the winner, and renormalizes the weights in the list. If it is wrong, consider-reward shifts the weight to the loser.

The test machine is now in charge of both setting and rewarding the behavior. The new primitive 'find' searches the world for a colored puck, then if it is found, a reward (or lack of reward) is given based on whether the machine is attending to the monkey's hand or the test-board. The end-of-test action-pattern calls actions in sequence from two different behaviors — the monkey's sequence learner learns, then the test box records and resets. Educated-grasp is now a method on sequence-learner; it does visual examination, evaluation, and the grasp.

The above is just the first two steps of the development of a learning version of Harris' model. This example demonstrates how functionality can be easily added, and shifted between plans and behaviors. The second model above converges to a correct solution within 150 trials, with significant-difference set to .08, and weight-shift to .06. We have additional forthcoming results showing that adding another layer of learning (the select-avoid rules) results in further characteristics typical of primate learning.

The sequence-learning task is interesting because it illustrates not only the interaction between reactive plans and modular behaviors, but also begins to model how a behavior might be designed to learn a reactive plan. However, the BOD methodology is not constrained to modeling only a single agent. We are currently modeling conflict resolution in primate colony social interactions in collaboration with Jessica Flack of Emory University. In this work, we use different processes to model different agents. Each agent has its own instance of the behavior objects and its local copy of a POSH plan.

7 Conclusions

Software engineering is the key problem of developing complete agents. The advances made due to the reactive and behavior-based movements come primarily because they trade off slow or unreliable on-line processes of search and learning for the one-time cost of development, and by emphasizing the use of modularity. These are not reasons to fully abandon learning and search, but they are reasons to use it only in constrained ways likely to be successful.

It is easiest to design action selection if one exploits the sequencing and prioritization skills of programmers. We have shown how sequences can be adapted into powerful reactive plans. One of the components we consider critical to successful reactive planning, the BRP, is present to our knowledge in only one architecture with a reasonably large user base, TR [Nilsson, 1994]. Most other architectures either only allow for reactivity *between* plans, or don't allow for structured plan elements at all. We have also created a mapping between the three layer architectures that dominate complete agent re-

search today, and the original behavior-based architectures, which are more like today's MAS architectures. We suggest that the primitive elements typical of the lowest layer of a three-layer architecture should interface to semi-autonomous behavior modules, which are comparable to high-level processes in some three-layer architectures. This allows the basic actions coordinated by the reactive plans direct access to appropriately processed state necessary for informing perception and determining the parameters of actuation.

References

Ronald Arkin. Integrating behavioral, perceptual and world knowledge in reactive navigation. *Robotics and Automation*, 6(1):105–122, 1990.

Arvind K. Bansal, Kotagiri Ramohanarao, and Anand Rao. Distributed storage of replicated beliefs to facilitate recovery of distributed intelligent agents. In Munindar P. Singh, Anand S. Rao, and Michael J. Wooldridge, editors, *Intelligent Agents IV (ATAL97)*, pages 77–92, Providence, RI, 1998. Springer.

Scott Benson. *Learning Action Models for Reactive Autonomous Agents*. PhD thesis, Stanford University, December 1996. Department of Computer Science.

Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

Joanna Bryson and Brendan McGonigle. Agent architecture as object oriented design. In Munindar P. Singh, Anand S. Rao, and Michael J. Wooldridge, editors, *The Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL97)*, pages 15–30, Providence, RI, 1998. Springer.

Joanna Bryson and Lynn Andrea Stein. Architectures and idioms: Making progress in agent design. In C. Castelfranchi and Y. Lespérance, editors, *The Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL2000)*. Springer, 2001. *in press*.

Joanna Bryson and Kristinn R. Thórisson. Dragons, bats & evil knights: A three-layer design approach to character based creative play. *Virtual Reality*, 2001. *in press*.

Joanna Bryson. Cross-paradigm analysis of autonomous agent architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 12(2):165–190, 2000.

Joanna Bryson. Hierarchy and sequence vs. full parallelism in reactive action selection architectures. In *From Animals to Animats 6 (SAB00)*, pages 147–156, Cambridge, MA, 2000. MIT Press.

Richard W. Byrne and Anne E. Russon. Learning by imitation: a hierarchical approach. *Brain and Behavioral Sciences*, 21(5):667–721, 1998.

David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

Luis Correia and A. Steiger-Garção. A useful autonomous vehicle with a hierarchical behavior control. In F. Moran, A. Moreno, J.J. Merelo, and P. Chacon, editors, *Advances in Artificial Life (Third European Conference on Artificial Life)*, pages 625–639, Berlin, 1995. Springer.

Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

Erran Gat. Three-layer architectures. In David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, chapter 8, pages 195–210. MIT Press, Cambridge, MA, 1998.

M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, Seattle, WA, 1987.

Kristian J. Hammond. Case-based planning: A framework for planning from experience. *The Journal of Cognitive Science*, 14(3), September 1990.

Mitch R. Harris and Brendan O. McGonigle. A modle of transitive choice. *The Quarterly Journal of Experimental Psychology*, 47B(3):319–348, 1994.

Chris Knight, Michael Studdert-Kennedy, and James R. Hurford, editors. *The Evolutionary Emergence of Language: Social function and the origins of linguistic form*. Cambridge University Press, 2000.

David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors. *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*. MIT Press, Cambridge, MA, 1998.

Pattie Maes. Situated agents can have goals. In Pattie Maes, editor, *Designing Autonomous Agents : Theory and Practice from Biology to Engineering and back*, pages 49–70. MIT Press, Cambridge, MA, 1990.

Brendan McGonigle and Margaret Chalmers. Are monkeys logical? *Nature*, 267, 1977.

Nils Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

Tony J. Prescott, Kevin Gurney, F. Montes Gonzalez, and Peter Redgrave. The evolution of action selection. In David McFarland and O. Holland, editors, *Towards the Whole Iguana*. MIT Press, Cambridge, MA, to appear.

Phoebe Sengers. *Anti-Boxology: Agent Design in Cultural Context*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1999.

Rogier M. van Eijk, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Generalised object-oriented concepts for inter-agent communication. In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents VII (ATAL2000)*. Springer, 2001.

Patrick Winston. Learning structural descriptions from examples. In Patrick Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill Book Company, New York, 1975.

David H. Wolpert. The lack of A priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

Reflective Negotiating Agents for Real-Time Multisensor Target Tracking

Leen-Kiat Soh and Costas Tsatsoulis
Information and Telecommunication Technology Center (ITTC)
Department of Electrical Engineering and Computer Science
The University of Kansas
2335 Irving Hill Road, Lawrence, KS 66045 USA
{lksoh, tsatsoul}@ittc.ukans.edu

Abstract

In this paper we describe a multiagent system in which agents negotiate to allocate resources and satisfy constraints in a real-time environment of multisensor target tracking. The agents attempt to optimize the use of their own consumable resources while adhering to the global goal, i.e., accurate and effective multisensor target tracking. Agents negotiate based on different strategies which are selected and instantiated using case-based reasoning (CBR). Agents are also fully reflective in that they are aware of all their resources including system-level ones such as CPU allocation, and this allows them to achieve real-time behavior. We focus our discussion on multisensor target racking, case-based negotiation, and real-time behavior, and present experimental results comparing our methodology to ones using either no negotiation or using a static negotiation protocol.

1 Introduction

We describe a negotiating agent approach to multisensor target tracking and distributed resource allocation in a real-time environment. Each agent controls a set of resources, and is motivated to use these resources to track targets appearing in its coverage area, and also to make the resources available to other agents in an effort to satisfy the global tracking goals. The act of balancing the local use of tracking resources and the global goal satisfaction increases the complexity of the problem. The agents have to incorporate real-time issues into their decision making process since global tasks and resources are bounded by time. Each agent in the system is autonomous, monitors its environment through a sensor, reacts to changes that it observes, and maintains its own knowledge bases. There is no hierarchical organization among the agents allowing the system as a whole to react to world events more quickly. Since there is also no centralized information shared by the agents, information can only be exchanged directly during negotiation sessions and only what is considered relevant and useful information is communicated. This increases the autonomy of each agent and consequently strengthens the system's robustness. Since there is no top-down coordination, our agents dynamically form temporary coalitions to perform a task, with each agent in the coalition using and also making available its resources.

The driving application for our system is multisensor target tracking, a distributed resource allocation and constraint satisfaction problem. The objective is to track as many targets as possible and as accurately as possible using a network of sensors. Each sensor has a set of consumable resources, such as beam-seconds (the amount of time a sensor is active), battery power, and communication channels, that each sensor desires to utilize efficiently. Each sensor is at a fixed physical location and, as a target passes through its coverage area, it has to collaborate with neighboring sensors to triangulate their measurements to obtain an accurate estimate of the position and velocity of the target. As more targets appear in the environment, the sensors need to decide which ones to track, when to track them, and when not to track them, always being aware of the status and usage of sensor resources.

The problem is further complicated by the real-time constraints of the environment and the fact that agents have to share physical resources such as communication channels and disk storage. For example, for a target moving at one foot per second, accurate tracking requires one measurement each from at least three different sensors within a time interval of less than 2 seconds. The real-time constraints force our agents to deal with issues such as CPU allocation (since speed of execution depends on it), disk space allocation, communication latency, and processing times. Finally, the environment is noisy and subject to uncertainty and error: messages may be lost, a sensor may fail to operate, or a communication channel could be jammed. Thus, in addition to improving autonomy, one is required to promote noise-resistance in agent reasoning, sensor control, and communications.

The sensors are 9.35 GHz Doppler MTI radars that communicate using a 900 MHz wireless, radio-frequency (RF) transmitter with a total of eight available channels. Each sensor can at any time scan one of three sectors, each covering a 120-degree swath. Sensors are connected to a network of CPU platforms on which the agents controlling each sensor reside. The agents (and sensors) must communicate over the eight-channel RF link, leading to potential channel jamming and lost messages. Finally, there is software (the "tracker") that, given a set of radar measurements, produces a possible location and velocity for a target; the accuracy of the location and velocity estimates depend on the quality and frequency of the radar measurements: as we mentioned, the target must be sensed by at least three radars within a two second interval for accurate tracking.

Our solution to the problem is to use *reflective, case-based, negotiating agents*. The agents are *reflective* since they are aware of their resources (including computational ones) and of how their actions and commitments affect these resources. They also integrate case-based reasoning (CBR) and negotiation to dynamically form target-tracking coalitions and to determine how resources should be shared and used. CBR allows the negotiation to adapt to the dynamically changing environment. Negotiation allows a bottom-up generation of an any-time solution. All agents are peers, each responsible for initiating and responding to negotiations. When an agent senses an event that it cannot solve on its own, it dynamically forms a coalition from a subset of its known neighbors. It then initiates negotiation requests to the members of the coalition and conducts 1-to-1 negotiations. This way, the common goal of target tracking is divided into subgoals by the initiating agent.

The integration of real-time, CBR, and negotiation is a unique and innovative approach to the solution of a general class of dynamic, distributed, time-bound, over-constrained resource allocation problems represented by our domain of real-time multisensor target tracking.

2 Agent Negotiation

In our system negotiation is used to allocate sensor and computational resources and to allow the agents to reach an agreement on tracking a target. An agent is connected to and controls a sensor, and is aware of its state and the status of the resources it controls. Each agent operates in one of three different modes: tracking, negotiating, or both.

In this paper we will not discuss in detail how an agent tracks a target, since the topic is only tangentially related to negotiating agents. In a few words, an agent polls its sensor at predefined time intervals and if a radar return is considered "interesting," it then follows the potential target for one second sending all radar measurements to the tracker (the software component that computes target location and velocity given a set of radar measurements).

Since accurate target tracking requires triangulation, an agent that finds a potential target must contact other sensor-controlling agents to ask for their help. Illuminating a target by a radar implies the use of consumable resources: first, the radar will have to abandon its own target tracking (if any) to accommodate the request; second, using the radar consumes battery power; third, sending the measurements to the tracker occupies one of the eight globally available communication channels; fourth, the simple act of communicating between agents and handling the cognitive cost of this communication requires CPU resources. Agents are *cooperative* and desire the completion of the high-level goal of target tracking. At the same time, they are *individualistic*, in that they want to preserve their resources for their own goal satisfaction, and they are also *reliable*, in that they do not easily break a resource commitment made to another agent. Consequently, when an agent requests the use of the resources of another agent, it needs to convince that agent that it has priority in the use of these resources. To do so our agents use *negotiation.*

Negotiation can be used by agents to perform problem solving and to achieve coherent behavior in a multiagent system. Agents can negotiate in a fully prescribed manner where the negotiating parties know exactly what each other's cost and utility functions are, or when such knowledge is learned during the first step of interaction in a negotiation [Kraus, 1997; Kraus *et al.,* 1995]. There are agents that negotiate using the unified negotiation protocol in worth-, state-, and task-driven domains where agents look for mutually beneficial deals to perform task distribution [Rosenschein and Zlotkin, 1994; Zlotkin and Rosenschein, 1996]. Agents can also conduct argumentation-based negotiation in which an agent sends over its inference rules to its neighbor to demonstrate the soundness of its arguments [Jennings *et al.,* 1998]. Finally, there are agents that incorporate AI techniques [Chavez and Maes, 1996; Laasri *et al.,* 1992; Zeng and Sycara, 1998] and logical models [Kraus *et al.,* 1998] into negotiation.

2.1. Negotiation Model

Our agents use a variation of the *argumentative negotiation model.* Traditionally, in argumentative negotiation, an argument is a representation of a sequence of inferences leading to a conclusion [Jennings *et al.,* 1998]. Since our agents are assumed to share the same reasoning mechanism, it is not necessary for them to exchange their inference model with their negotiation partners. In addition, we assume that an agent reasons rationally and in good faith and is cooperative.

Before describing our negotiation model in detail we introduce some terminology: an *initiator* or initiating agent is the agent that requires a resource and contacts another agent to start a negotiating session; a *responder* or responding agent is the one that is contacted by the initiator; a *persuasion threshold* is a value associated with each resource or percentage of a resource that indicates the degree to which an agent needs to be convinced in order to free or share a resource (alternatively, one can view the persuasion thresh-

old as the degree to which an agent tries to hold on to a resource). Finally, a *negotiation strategy* dictates how an agent should behave before the start of a negotiation process; it spells out the time allowed for the agent to complete the negotiation, what type of information to send over as arguments, how agreeable the responding agent should be, and so on.

After the initiator determines that it requires assistance in tracking a target, it establishes a set of negotiation partners (a *coalition*, discussed in the next section), and contacts them to start negotiating. The initiator sends to the responders a message requesting a resource and a time interval it needs this resource (the resource, for example, can be turning on a radar beam at time T). The responder determines if it can satisfy the request immediately (for example, if the radar beam is already on), if it cannot negotiate at all (it is too busy, implying that it has no CPU resources available or no more threads), or if it can negotiate.

If both agents determine that negotiation is possible, they establish a negotiation strategy (see section 2.3) and start negotiating. Each agent has a local view of the world based on its sensor information. The initiator attempts to convince the responder by sharing parts of its local information. For example, it may share with the responder the speed with which the target is traveling or the other tasks it is currently performing; the responder uses this data to "see through the initiator's eyes," and determine if its needs are more pressing than its own. The responder uses a set of domain-specific rules to establish whether the information provided by the initiator pushes it above a resource's persuasion threshold, in which case it would free the resource. For example, a target being tracked by an initiating agent that is already busy tracking another target is a more convincing argument than a target that is already being tracked by multiple sensors.

If the responder is not convinced by the evidential support provided by the initiator, it requests more information from the initiator. The initiator, guided by its negotiation strategy, sends over what it views as its most useful arguments first. The responder evaluates these new arguments and updates the evidence support. This process iterates until either the agents reach an agreement, in which case a resource or a percentage of a resource is freed, or the negotiation fails.

2.2. Coalition Formation

In order to negotiate, an initiating agent must identify a group of other agents that it can talk to. This group is a negotiating coalition and is established dynamically by the initiator. To become a member of a coalition an agent must, first, be known to the initiator, and, second, have the potential to provide useful resources.

An agent knows a subset of the agents in the multiagent system. Usually it knows the agents in its physical neighborhood, since they all control radars that cover a certain area. Since, as mentioned, the radars are sessile, an agent only needs to be told once who its physical neighbors are. To establish who can provide useful resources, the initiator calculates the velocity of the target it is tracking and establishes a potential future path that the target will follow. Next, the initiator finds the radar coverage areas that the path crosses and identifies areas where at least three radars can track the target (remember that tracking requires almost simultaneous measurement from at least three sensors). The agents controlling these radars become members of the negotiating coalition.

Since computational resources are limited, and negotiating consumes CPU and bandwidth, the initiator does not start negotiation with all members of the coalition, but first ranks them and then initiates negotiation with the highest-ranked ones. Ranking of the coalition members is done using a multi-criterion utility-theoretic evaluation technique. The evaluation criteria are:

1. the target's projected time of arrival at the coverage area of a sensor: there has to be a balance between too short arrival times which do not allow enough time to negotiate and too long arrival times which do not allow adequate tracking;
2. the target's projected time of departure from the coverage area of a sensor: the target needs to be in the coverage area long enough to be illuminated by the radar;
3. the number of overlapping radar sectors: the more sectors that overlap the higher the chance that three agents will agree on measurements, thus achieving target triangulation;
4. whether the initiator's coverage overlaps the coverage area of the coalition agent: in this case the initiator needs to convince only two agents to measure (since it is the third one), which may be easier than convincing three;
5. the success rate in previous negotiations between the initiator and the agent in the coalition: previous successes are an indicator that an agent is more willing to be persuaded to free resources (since all agents are collaborative this is an indirect indication that an agent is mostly idle or has more resources than it needs).

At the end of the evaluation all coalition members are ranked and the initiator activates negotiations with as many high-ranked agents as possible (there have to be at least two and the maximum is established by the negotiation threads available to the initiator at the time, since it may be responding to negotiation requests even as it is initiating other negotiations).

2.3. Case-Based Negotiation Strategy

As *negotiation strategy* we define the set of guidelines (or protocol) that govern the behavior of an agent during a particular negotiation. In contrast to other work in negotiation where the negotiating parties followed a predefined, static protocol, our agents dynamically establish a new strategy depending on their current state and the state of the world.

The goal is to situate a negotiation and to improve the chances of its success by taking into account the dynamically changing world state. This is accomplished by using CBR to select, adapt, and eventually learn negotiation strategies.

Since initiating a negotiation and responding to one are fundamentally different tasks, although still governed by the same methodology, each agent has two different case bases: one with strategies for initiating negotiations and one with strategies for responding to negotiation requests. Cases of both initiating and responding negotiation strategies have the same description, but different strategies. In the following we discuss the joint situation description of the two case types and then discuss the two types of strategies separately.

A case contains a description of a situation that allows an agent, using simple weighted matching, to establish similarity between the current situation and the cases in the case base. The situation describes the state of the agent (tasks it is performing, state of the radar, its battery, etc.), the state of the target (current location and speed, projected path, type, etc.), and the model of the potential coalition members (how many, the number that actually were used in negotiation, their capabilities, etc.) Since an agent is always aware of this information, it can match the current situation with the description of the cases in the case base, find the best match, and apply (after adaptation) the negotiation strategy in the case to the current negotiation task.

Each case also contains the negotiation strategy that was used in the past together with the outcome of the negotiation, such as: "offer accepted," "offer rejected," "ran out of time," or "ran out of resources." The strategy tells the agent how to conduct the negotiation. For the initiator the negotiation strategy consists of the following:

1. a ranking of the classes of information it should use as arguments: during a negotiation each agent attempts to minimize the number and length of messages sent, since with fewer messages the agents can avoid message loss due to communication failures, and reduce traffic among the agents. The agents want to send short messages as well since the transfer is faster and the bandwidth is constrained. Thus, it is important for an initiating agent to decide which information pieces are more important to send to the responding agent;
2. the time constraint: how long (in real time) the agent should be negotiating, since the target may leave the area;
3. the number of negotiation steps: a "step" is a complete negotiation communication act where the initiator sends arguments and the responder makes a counter-offer or requests more convincing arguments. Clearly the more steps that are allowed the higher the chance of reaching an agreement, but also the more time and resources are spent;
4. the CPU usage: more CPU resources for a negotiation mean faster negotiation, but also less CPU available for other tasks.

The responder has a slightly different negotiation strategy. It shares some elements of the initiator's protocol, specifically the time constraint, the number of negotiation steps, and the maximum CPU usage, but it also introduces two more parameters:

1. the power usage: this defines how much power the responder is willing to use to turn on its radar;
2. persuasion thresholds for resources: as already mentioned, each resource has a persuasion threshold associated with it which determines how difficult it will be to convince the responder to free the resource. The resources are radar sectors for performing frequency or amplitude measurements to track a target, CPU allocation, and usage of the RF communication channels. Discrete resources like turning on a radar, have a single valued persuasion threshold. Continuous resources like CPU, where a responder may agree to free a percentage of it, have a linear or an exponential function of evidence support, as persuasion thresholds (so, if an initiator convinces a responder by degree X, then the responder is willing to free N% of its CPU allocation; if it is convinced by degree (X+Y) it will be willing to free (N+M)% of its CPU, where $N = f(X)$, $M = f(X+Y) - f(X)$, and f is either a linear or an exponential function depending on the actual situation). We have chosen these two functions since they are easy to compute and represent two different conceding behaviors—the linear function has a uniform conceding rate whereas the exponential function models agents willing to concede quickly.

After a case has been selected and an old negotiation strategy has been retrieved, the agent adapts the strategy to best fit the current situation. Our adaptation technique uses two sets of rules: one that maps the differences between the case description and the current world state into strategy fixes, and a second one that uses the outcomes of the old negotiation strategy to guide its adaptation into a more potentially successful one. For example, if the current target is faster than the target in the case, then the agent reduces the negotiation time in its strategy. Or, if the old negotiation failed because the agents could not reach an agreement, then the agent may want to use fewer CPU resources and plan to spend less time on the negotiation, since it may fail again. Currently, we have 17 difference-driven and seven outcome-driven domain-specific adaptation rules. On average, each rule has two conditions to facilitate fast token matching and has about three conclusions such as "increase the number of negotiation steps by X," "decrease the time by Y," and so on.

Finally, when a negotiation strategy has been created, the agents engage in negotiation, as discussed in section 2.1. After the negotiation is concluded, the agents involved in it decide whether it is worthwhile to learn the strategy they

used, and to add it to their respective case base. An agent matches the new case to all cases in the case base and if it is significantly different from all of them it learns the new strategy by storing the case in the case base. By learning cases whose description differs sufficiently from the ones in the case base the agents attempt to improve their negotiation strategies by covering a larger part of the problem domain. If, though, the new case is not sufficiently different from the ones in the case base, the agent determines which one to keep: the new one or the old case which best matches the new one? To do so the agent attempts to increase the diversity of the case base by computing the sum of differences between that old case and the entire case base (minus the best matching old case) and the sum of differences between the new case and the entire case base (minus the best matching old case). If the second sum is greater than the first sum, then it replaces the old case with the new case. The heuristics we use to evaluate whether a new case should be learned or not are similar to the similarity evaluation performed during retrieval, with the additional evaluation of the solution parameters. Since learning is performed by the agent on-line, we have designed it to be of only linearly related to the number of cases in the case base. Consequently, the diversity measurement is between the new case and every case in the case base instead of between all case pairs. This allows us to improve the speed of the learning step when a new case comes in and to reduce the computational requirements.

There has been work in off-line learning of negotiation strategies using genetic algorithms [Matos *et al.*, 1998], but in our work learning is continuous and on-line.

3 Real-Time Reflective Agents

A fundamental concern in multisensor target tracking is the timeliness of the measurements: a radar must be active and illuminating an area when a target is passing through it and when other radars are measuring, too. This introduces real-time constraints to the sensor management by the agents: negotiations must be concluded within sufficient time to allow execution of sensing commands, or must be aborted to allow negotiation with other members of the coalition. To achieve real-time behavior the agents must be fully aware of the status of system-level resources and of the passage of time. This awareness defines a *real-time reflective agent*.

Our agents use the Real-Time Scheduling Services (RTSS) that reside on top of the KU Real-Time system (KURT) [Srinivasan *et al.*, 1998] that adds real-time functionality to Linux. First, the RTSS provides an interface between the agents and the system timers, allowing agents to: (1) query the OS about the current time; (2) ask the RTSS to notify them after the passage of certain length of time; and (3) ask the RTSS to ping them at fixed time intervals. This allows agents to know when to, for example, conclude a negotiation process or turn on a radar sector. Second, the agents may ask the RTSS to notify them when certain system-level events occur, such as process threads being activated, or communication messages going out or coming into the system. Third, the agents can ask the RTSS to allocate them a percentage of the CPU for each one of their threads (such as the ones controlling the radar and tracking or the ones used in negotiations) and to schedule this allocation within an interval of time. This way agents residing on the same computational platform can establish execution priorities and can control how fast an operation can be performed (clearly, more CPU scheduled in consecutive time intervals implies faster execution for a thread, leading to faster reasoning and negotiation).

The RTSS may be unable to perform such a CPU allocation and scheduling, if, for example, all available CPU resources are already occupied. Then, the requesting agent is notified and is also informed of which agents are using the CPU resources. This allows the agent to initiate a negotiation for CPU with the other agents. This is when the fourth function of the RTSS comes into play: an agent needs to know what percentage of its allocated CPU it is using, to be able to determine whether it is willing to give up part of its CPU allocation to a requesting agent. After agents have negotiated a new sharing of CPU resources they request a rescheduling of the allocations, and the RTSS dynamically performs it.

The RTSS allows agents to be full masters of all their resources, including system-level ones. Agents can negotiate about CPU and can simply cede part of their allocation to other agents. Knowledge of the passage of real time, of the occurrence of system-level events, and of CPU usage and load make our agents reflective and allow them to function effectively in a real-time domain.

Previous work in real-time AI fell under two general categories: (1) anytime algorithms [Dean and Boddy, 1988] where a solution to a problem can be incrementally refined and can be applied at anytime during the refinement process, and (2) multiple methods or approximate processing [Lesser *et al.*, 1988] where different approaches to a solution are available and can be combined.

4 Experimental Results

The reflective, real-time, case-based negotiating agents described in the previous sections have been implemented and tested using real sensors and targets moving in a physical environment. The agents exhibit all of the behavior described: they use CBR to select and adapt a negotiation strategy, use the RTSS to request CPU resources and to have time and system awareness, negotiate for radar use, and learn the new negotiation strategies they have developed. Most importantly, the agents achieve the high-level goal of the system: they track targets traversing an area covered by many radars.

Our experiments concentrated on evaluating whether negotiating agents can track targets better, and whether CBR results in better negotiation strategies. Our hypotheses

were, first, that negotiating agents can track targets better since they can coordinate radar measurements and achieve better triangulation, and, second, that negotiation using CBR will result in better tracking than using a static negotiation protocol, since CBR will allow adaptation of the strategy to the current situation. Our experiments support these hypotheses. In addition to the accuracy of tracking, we used communication as a measure of quality (length of messages, frequency of messages and message cost, i.e. length times frequency), since communication is an important bottleneck of scale up.

First we compared our system to a multiagent, sensor-controlling network where there is no communication between the agents, and where when a target appears in the coverage area of a sensor it is tracked. Next, we compared our case-based negotiating agents to a system where negotiation uses a predefined, static strategy. We selected the static strategy carefully to make sure it should be adequate for most cases.

In general, the results, summarized in figures 1-4, were very encouraging. The agents which used no negotiation sent almost 20% more messages but had almost 27% worse tracking accuracy than negotiating agents. The non-negotiating agents exchanged no messages, and only sent their radar measurements to the tracking software. Since there was no coordination of the measurements, there were too many messages sent to the tracker. On the other hand, we also found that such messages are short compared to arguments exchanged between agents during negotiation, resulting in lower message costs—the product of the average length and the total number of messages sent per second. Since there was no cooperation to triangulate measurements, the resulting accuracy was poor.

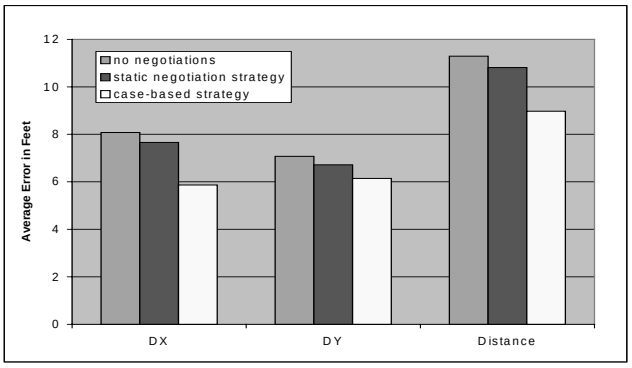


Figure 1: Tracking accuracy vs. agent behavior

The agents that used a static negotiation strategy fared worse than the ones that used a case-based, adaptive strategy. Specifically, the agents using a static protocol sent approximately 10% fewer messages (though with a slightly higher message cost) and had almost 18% worse accuracy than the case-based negotiating agents. The message cost is due to the fact that the case-based agents change the ranking of the arguments they communicate based on the situation; this leads to overall more effective communication acts. The accuracy is due to the fact that case-based agents adapt their negotiation to the current situation and have a higher chance of achieving agreement for resource allocation; on the other hand the static strategy agents failed to agree more often and this led to failure to perform the multiple, simultaneous radar measurements that are required for accurate tracking.

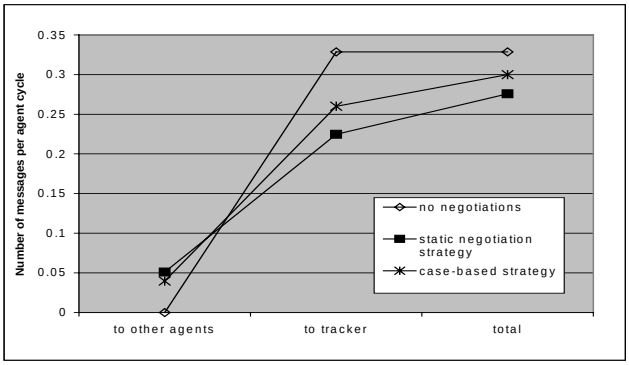


Figure 2: Number of messages to agents and to tracking software vs. agent behavior.

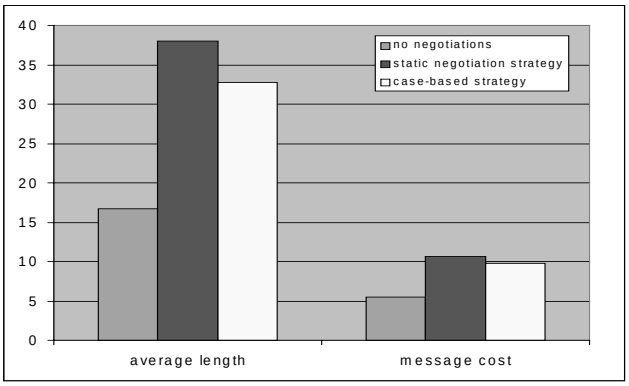


Figure 3: Message statistics vs. agent behavior. Message cost is the product of the average length and the total number of messages sent per second.

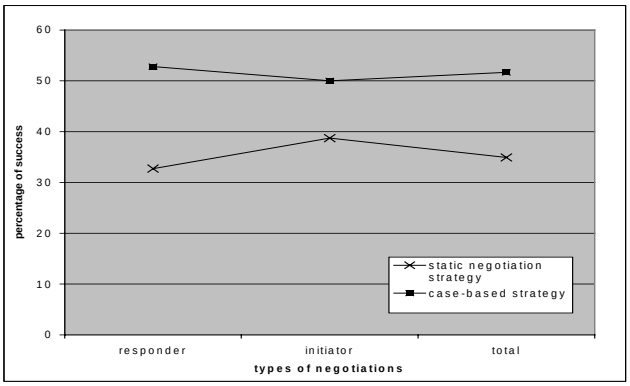


Figure 4: Percentage of successful negotiations vs. negotiation strategy type. A successful negotiation is one that completes with a deal between the two negotiating agents.

5 Conclusions

We have described a multiagent approach to distributed resource allocation problems, particularly to multisensor tracking of targets in a real-time environment. Our approach uses negotiations among agents to exchange information based on strategies retrieved using a case-based reasoning system. This allows the agents to learn negotiation strategies based on previous experiences, adapt to the current situations, and avoid repeating past failures. We have shown experimentally that CBR-based negotiations helped agents to negotiate more efficiently and more successfully, indirectly helping the agents track their targets more accurately. The agents in our system use real-time scheduling services to become reflective of the system-level resources they use and to be time-aware; this allows the agents to work in an environment of real-time constraints. Finally, we showed experimentally that reflective negotiating agents can track targets much better than agents that simply react to the presence of targets in their environment. The reflective nature of the agents allows them to schedule the precise time of measurement and also exchange computational resources, leading to faster and more efficient processing.

Acknowledgments

The authors would like to thank Kelly Corn, Will Dinkel, Jim Emery, Arun Gautam, Douglas Niehaus, Pete Prasad, and Huseyin Sevay for their work on the ANTS Project at the University of Kansas. The work described in this paper is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-0502. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

References

[Chavez and Maes, 1996] Chavez, A., and Maes, P. Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of 1st Int. Conf. on Practical Application of Intelligent Agents & Multi-Agent Technology*, 75-90.

[Dean and Boddy, 1988] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proc. of the Seventh National Conf. on Artificial Intelligence* (St. Paul, MN), August, 49-54,

[Jennings *et al*., 1998] Jennings, N. R., Parsons, S., Noriega, P., and Sierra, C. On argumentation-based negotiation. In *Proc. of Int. Workshop on Multi-Agent Systems* (Boston, MA).

[Kraus, 1997] Kraus, S. Beliefs, time, and incomplete information in multiple encounter negotiations among autonomous agents, *Annals of Mathematics and Artificial Intelligence* 20, 1-4, 111-159.

[Kraus *et al*., 1998] Kraus, S., Sycara, K., and Evenchik, A. Reaching agreements through argumentation: a logical model and implementation, *AI Journal* 104, 1-2, 1-69.

[Kraus *et al*., 1995] Kraus, S., Wilkenfeld, J., and Zlotkin, G. Multiagent negotiation under time constraints. *Artificial Intelligence* 75, 297-345.

[Laasri *et al*., 1992] Laasri, B., Laasri, H., Lander, S., and Lesser, V. A generic model for intelligent negotiating agents. *Int. J. of Intelligent & Cooperative Information Systems* 1, 291-317.

[Lesser *et al*., 1988] Lesser, V. R., Pavlin, J., and Durfee, E. Approximate processing in real-time problem solving. *AI Magazine* 9, 1, 49-61.

[Matos *et al*., 1998] Matos, N., Sierra, C., and Jennings, N. R. Negotiation strategies: an evolutionary approach. In *Proc. of Int. Conf. on Multiagent Systems (ICMAS'98)* (Paris, France), July 4-7, 182-189.

[Rosenschein and Zlotkin, 1994] Rosenschein, J. S., and Zlotkin, G. Designing conventions for automated negotiation, *AI Magazine* 15, 3, 29-46.

[Srinivasan *et al*., 1998] Srinivasan, B., Pather, S., Hill, R., Ansari, F., and Niehaus, D. A firm real-time system implementation using commercial off-the shelf hardware and free software. In *Proc. of the Real-Time Technology and Applications Symposium*, (Denver, CO).

[Zeng and Sycara, 1998] Zeng, D., and Sycara, K. Bayesian learning in negotiation, *Int. J. of Human-Computer Studies* 48, 125-141.

[Zlotkin and Rosenschein, 1996] Zlotkin, G., and Rosenschein, J. S. Mechanism design for automated negotiation, and its application to task oriented domains, *Artificial Intelligence* 86, 2, 195-244.

Stable Strategies for Sharing Information among Agents *

Rina Azoulay-Schwartz[1] **Sarit Kraus**[1,2]
[1]Department of Mathematics and Computer Science
Bar-Ilan University, Ramat-Gan, 52900 Israel
{schwart, sarit}@macs.biu.ac.il
[2]Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742

Abstract

Information sharing is important for different goals, such as sharing reputations of sellers among potential buyers, load balancing, solving technical problems, etc. In the short run, providing information as a response to queries is often unbeneficial. In the long run, mechanisms that enable beneficial stable strategies for information exchange can be found. This paper presents such mechanisms and specifies under which conditions it is beneficial to the agents to answer queries. We analyze a model of repeated encounters in which two agents ask each other queries over time. We present different strategies that enable information exchange, and compare them according to the expected utility for the agents, and the conditions required for the cooperative equilibrium to exist.

1 Introduction

In this paper, we consider the problem of information sharing among self motivated agents. Information sharing is necessary in environments where autonomous agents are required to solve problems, and additional information may improve their performance, i.e., reputation systems, load balancing, solving problems which require specialization, etc. Information sharing among agents in such environments is supposed to increase their average utility, since the cost of one agent to find an answer to a query is usually less than the utility derived by the agent who receives the response.

Research on information sharing among agents usually assumes that the agents are motivated to share information with each other and to help each other to find the best solution to their problems [Mor, 1996; Raub and Weesie, 1990; Zacharia, 1999]. This assumption does not hold in multi-agents environments, where each agent belongs to another owner, and wants to maximize its own utility. An agent when answering a query bears the costs of searching for the answer, and sending it to the questioner, and it may also bear indirect costs. For example, if the query is about the resource with the lowest load [Schaerf *et al.*, 1995], answering it may increase the load of the resource, and this can harm the responding agent that publicized this information. The responding agent does not receive any payment for its answer, since there is no mechanism to enable such a payment. Moreover, the value of an answer cannot objectively be evaluated, and payment for answers may cause the queries flow to be damaged only as a result of evaluation problems.

In fact, each agent would like to receive answers to its own queries, while ignoring queries directed to it. Thus, as we show in Section 1.1 below, it is clear that in equilibrium of a single interaction, no agent will answer any query. However, if the interactions are repeated, strategy profiles exist in which it is worthwhile for the agents to attempt to answer queries, since their long term utility will increase.

To simplify the problem, we analyze a model of repeated interactions in which two agents contact each other and ask queries repeatedly. This problem is different from the classical prisoner's dilemma [Fudenberg and Tirole, 1991] in respect to two main issues. First, the agents do not make their decisions simultaneously: in each interaction, one agent asks a query, and the decision is made by the second agent. Second, an agent, when attempting to answer a query, may fail to find an answer, and the questioner cannot know whether it did not receive an answer because the other agent ignored its query, or because the other agent failed to find an answer. In fact, the agent which has to answer may also return a negative message, saying it cannot find an answer. However, such a response is strategically equal in our model to not responding, since in that case, the questioner cannot know whether the agent really attempted to answer its query or not. We also assume that an agent cannot send a fictive answer, since such an answer will be revealed immediately. (e.g., information about a seller cannot be given if the informer does not know actual details about it. Technical help which is not useful will immediately be found to be worthless, etc.)

Other research conducted in DAI concerning repeated interactions, deals mostly with learning the best strategy to use in repeated interactions [Sen and Arora, 1997; Sandholm and Crites, 1995; Carmel and Markovitch, 1996; Freund *et al.*, 1995]. Sen and Arora developed and analyzed probabilistic *reciprocity schemes* as strategies to be used by self interested agents to decide whether or not to help other agents. The principle of reciprocity is that agents only help those agents

*This material is based upon work supported in part by the NSF under Grant No. IIS-9820657. Rina Azoulay-Schwartz is supported in part by the Israeli Ministry of Science.

who helped them in the past or can help them in the future. Their analysis and experiments show that reciprocal behavior improves the individual agent's performance in the long run over the selfish behavior. Our research also deals with repeated interactions among self interested agents, but we take the classic game theory approach of finding strategies that are in equilibrium. Chalasani at el [Chalasani *et al.*, 1998] developed a model where querying agents send queries to information agents. They designed a randomized symmetric strategy which minimizes the expected completion time of a query. However, they do not explain the motivation of an agent to use the symmetric strategy. In our research, we consider strategy profiles, which the agents are motivated to follow. We evaluate the expected utility of the strategy profiles, and the conditions required for this profile to be an equilibrium. We combine theoretical proofs with particular examples that demonstrate the behavior of the strategy profiles for particular parameters.

1.1 The one period interaction

Consider the following interaction of two agents, i and j: Agent j is ready to ask a query, and it can either send it to agent i or not. If it sends the query, then agent i can either attempt to answer the query or not. If agent i attempts to answer the query, then with a probability of p_i it will succeed in answering the query, but with a probability of $1 - p_i$ it will fail, where $0 \leq p_i \leq 1$. If agent j does not receive an answer, it does not know whether agent i attempted to answer it and failed, or whether it even tried.

Agent i, when attempting to answer a query, incurs an obligatory cost o_i, for searching for an answer. If it succeeds in finding an answer, then it incurs an additional cost of c_i, which contains the expenses of retrieving the answer (i.e., its total cost is $o_i + c_i$). If agent i does not attempt to answer the query at all, then it will have a utility of 0. The asking agent (agent j) obtains a utility of v_j only if it receives an answer. In any other case, its utility will be 0.

Consider the one period interaction in which agent B is ready to to send agent A a query. There are two pure equilibria for this interaction: in the first, agent B will send the query to agent A, but agent A will not attempt to answer it. Note that agent B still sends its query, since we assume that it incurs no cost for sending queries. In the second, agent B will not send the query at all. In both equilibria, the utility of both agents is 0. In this paper, we present strategy profiles to be used by agents participating in the repeated version of the above interaction. We prove that under certain conditions, responding to queries is in equilibrium, and improves the agents' expected utility.

In fact, the problem can be stated more generally. Agent i can ask agent j to perform any arbitrary action, rather than answer a query. The action is costly to agent j, and it may succeed or fail. However, if the required results of the action are not achieved, agent i cannot observe whether this happened because of a failure of the action taken by agent j, or since agent j did not even attempt to perform the action. The problem is different from the classical *repeated principal-agent* problem [Radner, 1985], since each agent has a role of *a principal* in a part of the interactions, and has a role of an *agent* in the other. In the rest of the paper we refer to query answering, although our results are also appropriate for the general problem.

1.2 The repeated interaction

In the repeated interaction, there are several occurrences over time of the single interaction described above. We consider an *alternating* queries model, in which agent A asks a query, then agent B, and vice versa. Time is discrete and is indexed by $t = 1, 2, ...$ If it is agent i's turn to ask a query, then the probability for it to have a query at a given time period is q_i, and this probability is known to both agents. Although the agents know the probability distribution of the queries appearance, they do not know the actual time when queries will appear. This means that at a given time, each agent does not know the exact time it will be ready to send its next query, or the time its opponent will send its next a query. If a query was asked, then the one period interaction occurs, and we assume that it takes one time period. We consider a discounted utility function, and denote the discount factor of the utility function δ, where $0 \leq \delta < 1$. We assume that interactions continue indefinitely. In fact, our model also suits situations where in each interaction, there is a positive probability that no more interactions will occur, as described in [Osborne and Rubinstein, 1990]. Finally, ω denotes a configuration vector: $\omega = (p_A, p_B, q_A, q_B, \delta, v_A, v_B, c_A, c_B, o_A, o_B)$ and Ω denotes the set of all possible configuration vectors.

In this paper, we suggest a trigger strategy equilibrium [Fudenberg and Tirole, 1991] to be used by the agents in the repeated interaction. Trigger strategies are appropriate for cases where the action performed by one agent is unobserved by the other one, and it yields an outcome that is observed by both agents. However, the same outcome may be the result of different actions, with different probabilities. In this type of equilibrium, an agent uses the outcome of its opponent's action in order to decide whether to behave cooperatively, or to punish its opponent, and apply the non-cooperative strategy.

2 A one-period observation model

In this section, we consider a punishment for each time an answer is not obtained from the opponent, though there are cases in which this was not deliberately caused by the opponent. In fact, using a trigger strategy profile causes the agents to attempt to answer each other's queries, thus increasing the agents expected utility with regards to the case where the equilibrium of the one period interaction is implemented. However, there are cases where agents are punished due to failure in answering queries. We begin this section by defining the trigger strategy profile.

We suggest that the agents use a trigger strategy profile which is based on three possible "phases": $Normal$, $Punish_A$ and $Punish_B$. In phase $Normal$, each agent attempts to answer the query of the other agent. In phase $Punish_i$, agent j ignores the queries of agent i, but if agent j asks a query, agent i will attempt to answer it. At the beginning, the agents are in phase $Normal$, and remain there providing each agent answers its opponent's query. Given phase $Normal$, whenever an agent i does not answer a query, they

switch to $Punish_i$. This punishment phase holds until agent i answers a query of agent j, in which case, they return to phase $Normal$. The above strategy profile promotes cooperation and information sharing. In the next section we reveal under which conditions it is an equilibrium.

$\mathcal{D}_i$ is the expected discount ratio from the time agent i asks a query, until the time agent j will be ready to ask a query. We proved that

$$\mathcal{D}_i = \delta q_i + \delta^2(1-q_i)q_i + ... = \frac{\delta q_i}{1-\delta(1-q_i)}. \quad (1)$$

Denote the present time t_0, the time when agent B asks a query t_B, and the time after t_B in which agent A sends a query to B t_A. Denote the overall expected discount ratio from t_0 until t_A, $\mathcal{D}$. We proved that

$$\mathcal{D} = \frac{\delta^2 q_A q_B}{(1-\delta(1-q_A))(1-\delta(1-q_B))} = \mathcal{D}_A\mathcal{D}_B \quad (2)$$

Symmetrically, we proved that $\mathcal{D}$ is also the expected discount in the case of punishing agent B. We proved that $0 \leq \mathcal{D}_i < 1$, and also that $0 \leq \mathcal{D} < 1$. The proofs of this proposition, as well as the other proofs, can be found in [Azoulay-Schwartz, 2001]. For space limitation, we do not present them here.

2.1 Expected Utility and Equilibrium Conditions

In this section, we specify the expected utility of the agents when they follow the strategies profile described above, and the conditions under which this profile is an equilibrium. First, we define the terms that will be used for these specifications.

Definition 2.1 *The following terms express the expected utility of the agents, from the present until infinity.*

- V_i *: the expected utility of agent* i *if it attempts to answer the query of agent* j *(whether it succeeds or not).*
- U_i*: the expected utility of agent* i *when it is agent* j*'s turn to answer* i*'s query (whether* j *succeeds or not).*
- F_i*: the expected utility of agent* i *as the agents move to phase* $Punish_i$ *(either since agent* i *ignored the last query, or because of agent* i*'s failure to answer it).*

Generally, we consider the expected utility and the trigger equilibrium condition of agent A. B's specifications can be detailed symmetrically. We consider an unrestricted horizon model, so the utility terms are defined recursively.

Attribute 2.1 *The values of* V_A*,* U_A*, and* F_A *are computed as follows.*

$$V_A = -o_A + p_A(-c_A + \mathcal{D}_A U_A) + (1-p_A)F_A \quad (3)$$
$$U_A = p_B(v_A + \mathcal{D}_B V_A) + (1-p_B)\mathcal{D}U_A \quad (4)$$
$$F_A = \mathcal{D}V_A \quad (5)$$

V_A is the expected utility of agent A from attempting to answer a query. It consists of the expected future utility when the attempt to answer succeeds, and the expected utility when it fails, with the corresponding probabilities for both events, and the obligatory cost o_A. In case of success, the utility of agent A consists of the cost of c_A, and of its utility from asking agent B a query (U_A), after an expected discount of $\mathcal{D}_A$. In case of failure, agent A's utility is F_A.

U_A is the expected utility of agent A when it asks a query. If agent B succeeds to answer agent A's query, then agent A receives an immediate utility of v_A, and the agent stays in state $Normal$, i.e., after a delay of $\mathcal{D}_B$, agent A will be required to answer agent B's query, with an expected utility of V_A. If agent B fails to answer the query, then after a delay of $\mathcal{D}$, it will be required to answer the next query of agent A, i.e., agent A's expected utility is U_A.

F_A is the utility of agent A when it does not answer the query of agent B. Agent A will be punished, and after an expected discount ratio of $\mathcal{D}$, again, it will be its turn to answer agent B's query, i.e., its expected utility will be V_A.

We proceed with identifying the conditions under which the trigger equilibrium exists. In particular, we use the strategy profile defined in the beginning of Section 2, and we specify the condition under which each agent prefers the trigger strategy over deviation and ignoring the queries, given that the second agent uses its trigger strategy. If the condition of each agent holds, then the trigger strategy profile is an equilibrium.

Lemma 2.1 *The trigger equilibrium of the one period observation strategy profile is an equilibrium if* $V_i \geq F_i$*, for* $i \in \{A, B\}$.

The above condition claims that whenever the utility of answering a query is higher for the agent than its utility from ignoring the query, a trigger equilibrium exists. In the following lemma, we found an explicit formula which defines V_A, by using formulas 3 - 5.

Lemma 2.2 *The expected utility of agent* A *when attempting to answer a query, can be formalized as follows.*

$$V_A = \frac{(1+\mathcal{D}p_B-\mathcal{D})(-o_A-p_Ac_A)+p_A\mathcal{D}_Ap_Bv_A}{(1+\mathcal{D}p_B-\mathcal{D})(1-\mathcal{D}+\mathcal{D}p_A)-p_A\mathcal{D}p_B} \quad (6)$$

Using lemma 2.1, and the definitions of F_A and F_B, we can progress with finding the explicit conditions for the existence of the trigger equilibrium. In fact, the condition of agent i can be displayed as a required ratio between v_i and $c_i + \frac{o_i}{p(i)}$ for agent $i \in \{A, B\}$.

Lemma 2.3 *If the agents are risk-neutral, then the one period observation strategy is an equilibrium for agents A and B, if the following condition holds for both* $i = A, j = B$ *and* $i = B, j = A$.

$$\frac{v_i}{c_i + \frac{o_i}{p_i}} \geq \left(\frac{1-\delta+\delta q_i}{\delta q_i p_j} + \frac{\delta q_j(p_j-1)}{p_j(1-\delta+\delta q_j)}\right) \quad (7)$$

Using the above lemmas, important properties can be identified, concerning the strength of the equilibrium and the influence of the configuration parameters on the conditions of the equilibrium and on the agents expected utility. We present our conclusions in the following section.

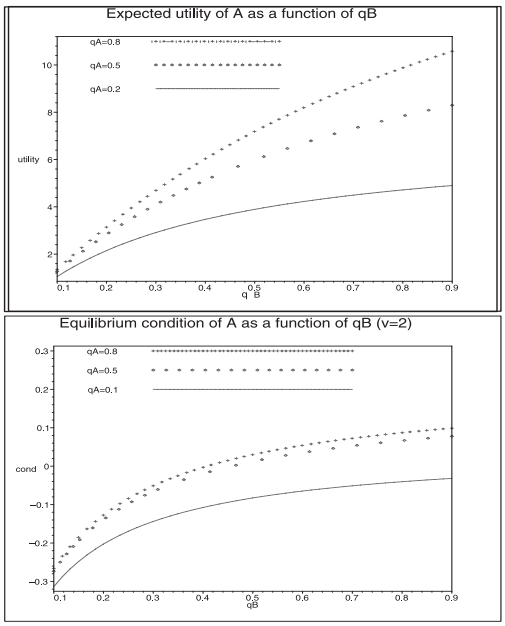


Figure 1: The influence of q_A and q_B: Up: the expected utility V_A. Down: the trigger equilibrium condition of agent A ($v_A = 2$; the equilibrium does not hold for all cases indicated by $y < 0$.)

2.2 Properties of the Expected Utility and of the Equilibrium Conditions

In this section, we study the existence of the trigger equilibrium and the agents' expected utility V_A. The value $V_A - F_A$ of lemma 2.1 should be non-negative for the equilibrium to exist. As this value increases, the trigger equilibrium exists for a larger set of configurations. Some of the conclusions are proved formally, while others are demonstrated for a particular configuration of parameters $\omega = (q_i = 0.1, p_i = 0.5, \delta = 0.9, c_i = 1, o_i = 0.1, v_i = 10)$.

Lemma 2.4 *As v_A, p_A or p_B increases, and as o_A or c_A decreases, the expected utility of each agent increases, and the trigger equilibrium holds for more configurations.*

The above conclusion is intuitive, since as the benefits an agent obtains from answering queries, v_A, increases, the utility of agent A increases, and it is more worthwhile for it to answer queries. It is also expected that the direction of influence of o_A and c_A will be opposite: as they increase, attempting to answer queries is more costly, so the utility, as well as the tendency of agent A to answer queries, decreases. Similarly, as p_A increases, V_A increases, since if agent A succeeds to answer more queries, its utility increases. As p_B increases, agent B succeeds to answer more queries of agent A, and agent A's utility increases (more cases where a utility of v_A is obtained), as well as its willingness to answer B's queries.

As δ increases, the expected utility of agent A also increases, as well as its tendency to attempt to answer agent B's query. The reason being that agent A bears present costs in order to achieve future benefits. Thus, as the discount of time decreases, the weight of the future benefits increases, and this causes the utility to increase, and the tendency to answer queries to increase too.

Figures 1-up and 1-down show that as q_A or q_B increase, the expected utility of agent A increases, as well as its tendency to follow the trigger strategy. As q_A increases, agent A is supposed to ask queries more frequently, so its utility from receiving answers increases. Thus, it is more beneficial for it to answer others' queries, since this will enable it to receive answers to its own queries. Thus, its tendency to attempt to answer queries, and its expected utility in this situation, increase with q_A. The influence of q_B is not intuitively clear. On the one hand, as q_B increases, agent B will ask a query more often, and this causes future costs for agent A. On the other hand, since the agents alternate in asking their queries, more frequent queries of agent B will cause agent A to also ask queries more often, and this may improve its utility, and its motivation to answer agent B's queries. In fact, Figure 1-up demonstrates that the influence of q_B is positive, and is similar to the influence of q_A. In order to check this phenomenon, we created 50,000,000 random configurations in which equilibria existed. In all these configurations the influence of q_B was positive. In [Azoulay-Schwartz, 2001] we considered the influence of q_B in situations where queries are not alternating, but at any given time, each agent can send a query. In these situations the influence of q_B is negative: as agent B is supposed to have more queries, the expected utility of agent A decreases.

To summarize, we have shown the influence of several parameters on the expected utility of agent A, and on its willingness to attempt to answer queries. Symmetric conclusions hold for agent B's utility and its trigger equilibrium condition. In fact, we can see that as the factors change in a direction that increases the utility of the agent, it will be more motivated to attempt to answer its opponent's queries. This conclusion does not hold for the situation of Section 3, as we change the length of the history that is taken into consideration.

3 A model with n periods observation

The trigger strategy is composed of punishment also in situations when the punished agent behaves cooperatively and follows its trigger strategy. This is unfair, and reduces the expected utility of the agents. Thus, we tested different strategy profiles where punishment is used, but more rarely. In this section, we consider a model, in which the n last periods are observed by the agent in order to decide whether to answer a query of its opponent or not. We consider two variations of the n periods observation model. In Section 3.1- 3.2, an agent is punished after n consequent queries with no answer by this agent. In Section 3.3, punishment is implemented after k unanswered queries, out of the n last queries to that agent.

In addition, in [Azoulay-Schwartz, 2001] we also considered a mixed strategy profile in which for some histories, an agent i will randomly decide whether or not to attempt to answer a query. There are two situations in which a mixed strategy can be considered. (a) in a punishment phase, where

agent i is allowed to punish agent j; (b) in the Normal phase, when agent i is supposed to answer j's query. We proved that a mixed strategy profile in a punishment phase (case (a)) is not stable. A mixed strategy profile may be stable in the *Normal* phase (case (b)), but we proved that its conditions are equivalent to those of the corresponding pure strategy, while the expected utility of the agents when using a mixed strategy profile, is lower than their expected utility when using the equivalent pure strategies. Thus, mixed strategies are not recommended for use in our model. In [Azoulay-Schwartz, 2001] we specify the model details and we prove our claims, but for space limitation, we do not specify these details here.

The results of the $n-1$ last events when agent i was required to answer queries is denoted h_i. The history of agent i is composed as follows: $h_i = (h_i(n-1), ..., h_i(2), h_i(1))$, where $h_i(1)$ represents the last event of a query sent to agent i. $h_i(k) = 0$ if the k's last query to agent i received no answer, and $h_i(k) = 1$ if the k's last query was answered by agent i. The term $h = (h_A, h_B)$ contains the $n-1$ last events with respect to the queries that agent A received, and the $n-1$ last events with respect to queries that agent B received. In particular, the notation $((1, ..., 1), (0, ..., 0))$, indicates a history of $n-1$ consequent successful answers of agent A, and $n-1$ consequent queries to agent B, with no response. Concatenating a new event to h_i, $h_i << new\_event$, means deleting the oldest event in h_i, and adding a new event to h_i. Finally, the function $zero(h_i)$ returns true if all the events in h_i are unanswered queries. Using these notations, we proceed with describing and analyzing both variations of the n-periods model.

3.1 Equilibrium with punishment after n failures

In this section, we analyze a model in which punishment of an agent is performed after n consecutive events of queries with no responses. The strategies and phases of the n periods model are defined as in Section 2, but moving from phase *Normal* to phase $Punish_i$ will occur only after n consequent queries with no response from agent i. A strategy is an ***n periods trigger strategy*** if it tells each agent i to answer queries of its opponent j, unless the last n queries sent to j received no answer. In this case, agent i ignores the queries of agent j, until it receives an answer from agent j to a query. Denote by $\Omega_n \subseteq \Omega$ the set of all $\omega \in \Omega$, such that the pair of n-periods strategies is an equilibrium given combination ω.

Assuming that both agents use their n-periods strategies, $V_A^{n,h}$ is the expected utility of agent A, when it obtains a query from agent B. Similarly, $U_A^{n,h}$ is the expected utility of agent A, when it waits for an answer from agent B. Suppose that the agents are in state $Normal$.

$$suc_A(h) = (-c_A + \mathcal{D}_A U_A^{n,(h_A<<1,h_B)})$$

is the expected utility of agent A from successfully answering a query of agent B. It includes the cost c_A, and the expected utility of asking a query after a delay of $\mathcal{D}_A$. Denote by

$$fail_A(h) = \mathcal{D}_A U_A^{n,(h_A<<0,h_B)}$$

the expected utility of agent A from a failure to answer agent B's query, if this didn't cause an immediate punishment. It includes an expected utility of asking a query after a delay of $\mathcal{D}_A$, but the failure is noted in h_A, and may cause a future punishment, if there will be future consequence failures. Finally,

$$pun_A(h) = \mathcal{D}V_A^{n,(h_A<<0,h_B)}$$

is the expected utility of agent A from a punishment. After a delay of $\mathcal{D}$, agent A will be expected to answer agent B's query. The expected utility of agent A when required to answer a query, denoted $V_A^{n,h}$, is defined as follows:

$$\begin{cases} -o_A + p_A \cdot suc_A(h) + (1-p_A)pun_A(h) & zero(h_A) = true. \\ -o_A + p_A \cdot suc_A(h) + (1-p_A)fail_A(h) & otherwise \end{cases}$$

Since agent A attempts to answer the query, it bears a cost of o_A. With a probability of p_A it will succeed in answering the query, and then its expected utility is $suc_A(h)$. With a probability of $1-p_A$, it will fail, and this will be noted in its history. If the current history of agent A includes only zeroes, then $Punish_A$ is reached, and the expected utility of agent A is $pun_A(h)$. Otherwise, its expected utility is $fail_A(h)$.

Similarly,

$$suc_B(h) = v_A + \mathcal{D}_B V_A^{n,(h_A,h_B<<1)}$$

is the expected utility of agent A when agent B succeeds answering its query,

$$fail_B(h) = \mathcal{D}_B V_A^{n,(h_A,h_B<<0)}$$

is A's utility when agent B fails to answer A's query, but punishment of B is not required, and

$$pun_B(h) = \mathcal{D}U_A^{n,(h_A,h_B<<0)}$$

is A's utility when punishing agent B is required. Using the above, the expected utility of agent A, when it forwarded a query to agent B, given n and h, denoted $U_A^{n,h}$, is defined as follows:

$$\begin{cases} p_B \cdot suc_B(h) + (1-p_B)pun_B(h) & zero(h_B) = true. \\ p_B \cdot suc_B(h) + (1-p_B)fail_B(h) & otherwise \end{cases}$$

With a probability of p_B, agent B will succeed in answering, and agent A's expected utility will be $suc_B(h)$. With a probability of $1-p_B$, agent B will fail to answer agent A's query. In this case, if punishment is required, then the expected utility of agent A is $pun_B(h)$. Otherwise, its expected utility is $fail_B(h)$.

For the expected utility calculation, the agent has to use an algorithm, based on the formulas of $V_A^{n,h}$ and $U_A^{n,h}$. In fact, these formulas depend on each other. In order to implement the calculation, a predefined depth (number of future periods) should be taken into consideration. A divide and conquer algorithm, or a dynamic programming algorithm, can be used in order to calculate the values of the formulas, given the required number of future periods.

3.2 Properties of the n-periods model

In the following, we analyze some important properties of the n-periods history model. In particular, we test the influence of n on the expected utility of the agents and on the conditions required for the existence of an n-periods strategy equilibrium. We start with an auxiliary claim.

Lemma 3.1 *Consider a trigger equilibrium based on the n periods strategy profile. The equilibrium will exist, if it is worthwhile for agent A to attempt to answer agent B after a history of $((1,...,1),(0,...,0))$, and it is worthwhile for agent B to attempt to answer agent A after a history of $((0,...,0),(1,...,1))$.*

After a history of $((1,...,1),(0,...,0))$, a future punishment of agent A due to current ignorance of a query has the lowest probability after the longest delay. Thus, if it is still worthwhile for A to hold the equilibrium strategy given this history, it will be worthwhile for it to do so after any other history. Similarly, if it is worthwhile for agent B to hold the equilibrium strategy given a history of $((0,...,0),(1,...,1))$, then it will be worthwhile for it to do so after any other equilibrium. Based on lemma 3.1, in order to determine whether an n-periods equilibrium exists or not, we only need to consider the history $((1,...,1),(0,...,0))$ of agent A, and the history $((0,...,0),(1,...,1))$ of agent B. In the following theorem, we prove that the set of configurations for which the n-periods equilibrium exists reduces as n increases, i.e., the trigger equilibrium exists more rarely as n increases.

Theorem 3.1 *For each $n \in N$, $\Omega_{n+1} \subset \Omega_n$. Moreover, for each $\omega \in \Omega$, $n \in N$ exists, such that $\omega \notin \Omega_n$, but for each $0 < n' < n$, $\omega \in \Omega_{n'}$.*

The motivation in the above theorem is that as n increases, the probability of punishment because of a present disregard of a query, becomes lower, and the time when this punishment will be used becomes more distant. Thus, there are more combinations for which the threat on an agent is not strong enough. The above theorem provides a simple rule for finding the optimal strategy profile for a given configuration. In fact, if an n-periods equilibrium does not exist, the agents should reduce n, until they obtain n' for which n'-periods equilibrium does exist. They should find the largest possible n', since, as proven in the following theorem, increasing n increases the expected utility of the agents.

Theorem 3.2 *For each $n \in N$, for each ω, such that $\omega \in \Omega_n \bigcap \Omega_{n+1}$, and for each history h, $V^{n+1,h} > V^{n,h}$.*

We demonstrate our main conclusions in Figure 2, for a particular configuration of parameters ($c_i = 1, o_i = 0.1, v_i = 100, p_i = 0.5, \mathcal{D}_i = 0.9$). The figure demonstrates that as n increases, $V_A^{n,h}$ increases too, as proven in lemma 3.2, but the increase is not linear: the increment level decreases as n increases. However, as proven in lemma 3.1, as n grows, the set of appropriate configuration values becomes smaller, and this is demonstrated in the lower dotted curve, which shows the difference between the expected utility of agent A if it attempts to answer a query after history $((1,...,1),(0,...,0))$, and its utility if it ignores the query. If the difference is positive, then an n-periods equilibrium exists, as was proven in lemma 3.1. It is also clear that as the difference increases, the n-periods equilibrium will exist for a larger set of parameters. As can be seen in the figure, the trigger equilibrium does not exist for n values higher than 6. This limit will be different for different parameter values, but the conclusion is clear. There is a trade off between the expected utility and the existence of a trigger equilibrium: as n increases, the expected utility of the agents decreases, while the trigger equilibrium exists for a smaller set of configurations.

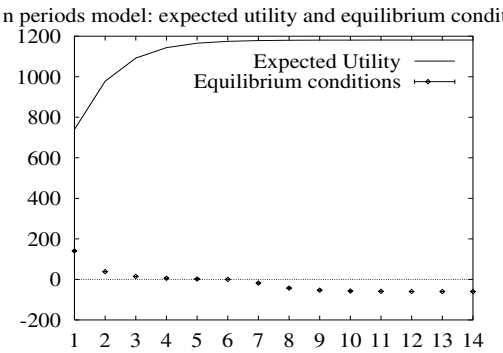


Figure 2: n-periods model: expected utility and trigger equilibrium condition as a function of n

The conclusion from this section is that given a parameters configuration, ω, the agents can decide about the optimal n to be used. In fact, they will choose the largest n for which a trigger equilibrium still exists, i.e., it is still beneficial for agent A to answer queries given history $((1,...,1),(0,...,0))$, and it is still beneficial for agent B to answer queries given history $((0,...,0),(1,...,1))$. Testing these conditions can be done by using a computation method based on the formulas of $V_A^{n,h}$ and $U_A^{n,h}$, as described in Section 3.1.

3.3 Punishment after k unanswered queries out of n

In this section, we consider a model, in which n periods of history are considered, but it is enough to observe $k \leq n$ unanswered queries of an agent, in order to decide to punish this agent. The model considered in Section 3.1- 3.2 is a special case of this model, with the restriction $k = n$. We denote $\Omega_{k,n}$ to be the set of configurations for which the strategy profile of punishment after k unanswered queries out of n, is an equilibrium. The next theorem summarizes our results concerning the influence of k on the agents' expected utilities and the existence conditions of the trigger equilibrium.

Theorem 3.3 *If $k1 < k2 \leq n$, punishment after $k2$ unanswered queries over n observed queries, achieves a higher expected utility than punishment after $k1$ unanswered queries over n observed queries, but $\Omega_{k2,n} \subset \Omega_{k1,n}$.*

The above theorem is intuitively clear, since as more unanswered queries are required in order to punish, then punishment is used more rarely, and this increases the agent's expected utility, while causing deviation to be beneficial in more situations. In order to check the influence of different values of n, we developed an algorithm based on dynamic-programming to compare the expected utility of agent A and its equilibrium condition for different strategy profiles, based on different n and ks. The results are presented in Figure 3, for: $\omega = (c_i = 1, o_i = 0.1, v_i = 20, p_i = 0.5, \mathcal{D}_i = 0.9)$. As proved in lemma 3.3, we can see that as k increases, the expected utility of agent A increases, while the equilibrium condition is weaker. However, the figure also demonstrates

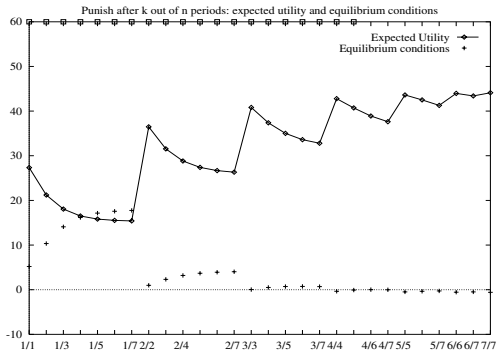


Figure 3: Punishment after k failures out of n: expected utility and trigger equilibrium conditions as a function of k/n

that as n increases, the expected utility of agent A decreases, while its condition becomes stronger. The intuition behind this result is that as the size of history observed for punishment increases from n to $n+1$, while keeping k fixed, a present ignorance of a query may cause a future punishment with a higher probability, since punishment can be utilized also due to a failure at $n+1$. Thus, the equilibrium conditions are expected to hold more frequently for larger n's. On the other hand, an increase of n will reduce the expected utility of the agent when it is required to answer a query, since failure to answer the query, will cause a punishment with a higher probability.

To summarize, we can see that different values of n and k may yield different values of expected utility and their particular value determines the existence of the trigger equilibrium. Thus, given a configuration of parameters, the agents have to find the pair of n and k for which the trigger equilibrium exists, (i.e., it is beneficial for both agents to follow the equilibrium strategies) and to choose a pair (n,k) from among them. Since there is no clear rule of choosing the optimal k and n, the agents should search all valid combinations of n and k to complete this task.

In fact, there may be situations in which each agent will prefer a different pair (n,k) due to different parameters values of the different agents. However, the agents can determine a rule of how to choose (n,k), such as, maximizing the average expected utility of them, or maximizing the product of the expected utility, etc. In the example demonstrated in Figure 3, the pair $k=n=3$ maximizes the expected utility of both agents, while the trigger equilibrium still exists. Thus, the agents should choose the equilibrium based on this pair.

4 Conclusion

In this paper, we present the problem of sharing information among self motivated agents. An agent receives queries and decides whether or not to attempt to answer them. First, we introduced the *one-period model*, in which each agent observes the last history event of its opponent in order to decide whether or not to answer it. Second, we introduced the model of punishing an agent after n unanswered queries. We found that as n increases, the expected utility of the agents increases, while there are more situations in which a trigger equilibrium does not exist. We also considered the general case, where punishment is implemented after k unanswered queries out of n queries, and we checked the influence of changing n and k.

In conclusion, we found that different punishment-based strategy profiles can be appropriate to attain responses in situations where attempting to answer queries is costly, and may result in success or failure. These profiles are stable, and increase the expected utility of the agents. Moreover, given a specific configuration, the agents may choose a strategy profile which maximizes the average, or product, of their expected utility, while a trigger equilibrium still exists. Additional variations of the model discussed in this paper are studied in [Azoulay-Schwartz, 2001]. In that paper we consider a model where at each time, each of the agents may have a query. In future work, we intend to consider situations where different parameters (n and k) are used by different agents, and also to consider a model where an agent can send its query to more than one agents.

References

R. Azoulay-Schwartz. *Protocols, Strategies and Learning Techniques for Reaching Agreements More Effectively*. PhD thesis, Bar Ilan University, 2001. forthcoming.

D. Carmel and S. Markovitch. Learning models of intelligent agents. In *Proc. of AAAI-96*, pages 62–67, 1996.

P. Chalasani, S. Jha, O. Shehory, and K. P. Sycara. Strategies for querying information agents. In *Cooperative Information Agents*, pages 94–107, 1998.

Y. Freund, M. Kearns, D. Ron Y. Mansour, R. Rubinfeld, and R. Schapire. Efficient algorithms for learning to play repeated games against computationally bounded adversaries. In *FOCS-95*, pages 332–341, 1995.

D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

Y. Mor. Computational approaches to rational choice. Master's thesis, Hebrew University, Jerusalem, Israel, 1996.

M. J. Osborne and A. Rubinstein. *Bargaining and Markets*. Academic Press Inc., San Diego, California, 1990.

R. Radner. Repeated principal agents games with discounting. *Econometrica*, 53:1173–1198, 1985.

W. Raub and J. Weesie. Reputation and efficiency in social interactions: An example of network effects. *American Journal of Sociology*, 96 (3):626–654, November 1990.

T. W. Sandholm and R. H. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, pages 147–166, 1995.

A. Schaerf, Y. Shoham, and M. Tennenholtz. Adaptive load balancing: A study in multi-agent learning. *JAIR*, 2:475–500, 1995.

S. Sen and N. Arora. Learning to take risks. In *AAAI-97 Workshop on Multiagent Learning*, pages 59–64, July 1997.

G. Zacharia. Collaborative reputation mechanisms for online communities. Master's thesis, MIT, Sept 1999.

CAST: Collaborative Agents for Simulating Teamwork

John Yen, Jianwen Yin, Thomas R. Ioerger,
Michael S. Miller, Dianxiang Xu, and Richard A. Volz
Department of Computer Science
H. R. Bright Building
Texas A&M University
College Station, TX 77843-3112, USA
{yen, jianweny, ioerger, mmiller, xudian, volz}@cs.tamu.edu

Abstract

Psychological studies on teamwork have shown that an effective team often can anticipate information needs of teammates based on a shared mental model. Existing multi-agent models for teamwork are limited in their ability to support proactive information exchange among teammates. To address this issue, we have developed and implemented a multi-agent architecture called CAST that simulates teamwork and supports proactive information exchange in a dynamic environment. We present a formal model for proactive information exchange. Knowledge regarding the structure and process of a team is described in a language called MALLET. Beliefs about shared team processes and their states are represented using Petri Nets. Based on this model, CAST agents offer information proactively to those who might need it using an algorithm called DIARG. Empirical evaluations using a multi-agent synthetic testbed application indicate that CAST enhances the effectiveness of teamwork among agents without sacrificing a high cost for communications.

1 Introduction

Teamwork has been the focus of a great deal of research, spanning diverse disciplines from business management to psychology [Ilgen et al., 1993]. There are many different types of teams, from those that are hierarchical to those that are more egalitarian. Some teams have fixed, clearly-defined roles, while others allow for dynamic re-allocation of tasks and responsibilities on the fly. Measuring the performance of a team can involve both "outcome" as well as "process" measures [Cannon-Bowers and Salas, 1997].

Several computational models of teamwork have been developed for producing cooperative behavior among intelligent agents. The fundamental aspect of a team that distinguishes them from just a group of interacting agents is that they share common goals. In the BDI framework [Rao and Georgeff, 1991; Wooldridge and Jennings, 1995], the mental state of having a team goal has been characterized in terms of *joint intentions* [Cohen and Levesque, 1991, Jennings, 1995]. Tambe [Tambe, 1997] and his STEAM group have shown how these mental states can be established and maintained through communication protocols. Another framework for modeling teams is through *SharedPlans*, via intentions to do certain steps together [Grosz and Kraus, 1996]. These approaches have been shown to be effective for simulating teamwork in a wide range of agent-only environments [Jones et al, 1999; Tidhar et al., 1998; Stone and Veloso, 1999].

However, these existing multi-agent teamwork models have not been designed for supporting mixed human/agent teams. Having humans in the loop places an additional constraint on the agents, such that they must interact with teammates in a natural way, e.g. by exchanging only the most important information necessary for coordination, without excessive or redundant communication. Efficient teamwork relies heavily on information sharing, especially in dynamic environments, but it must be done judiciously not to overwhelm the human participants with message passing. The key is to try to supply only the most *relevant* information, and this requires reasoning about their goals and responsibilities on the team.

Motivated by this observation, we have developed CAST (Collaborative Agents for Simulating Teamwork), a multi-agent architecture that simulates and supports teamwork. While our ultimate goal involves supporting both humans and agents, in this paper we will focus on reducing communication with software agents (in preparation for inclusion of humans) as a first step. Our architecture provides a mechanism for building virtual teams using software agents. We will first give an overview about the goals and issues addressed by CAST. A formal

foundation for proactive information exchange in CAST is then introduced. This is followed by a description of the major components and features of the CAST architecture. An empirical evaluation is used to assess the effectiveness of teamwork simulated in CAST. Finally, we summarize the main contribution of the work.

2 CAST Overview

CAST is designed to achieve two goals. First, it aims to model effective teamwork by capturing both *team structures and teamwork processes*. A well-defined team structure is based on specifying pre-defined roles and the responsibilities associated with them. A well-defined teamwork process specifies goals, strategies, and plans for accomplishing the team's goal. The common prior knowledge about the structure and the process of the team enables members of the team to develop an "overlapping shared mental model," which is the source for team members to reason about the states and the needs of others.

One of the major criticisms of other approaches toward the goal above has been that they lack flexibility for dealing with dynamic environments. Hence, another equally important goal is to enable agents in a team to have *flexibility* in adapting to changes in the environment. More specifically, assignment of responsibilities to suitable agents needs to be adapted to the current state of the world and the team.

While both goals are desirable, they conflict with each other. The emphasis on predefined team structures and processes often reduces the flexibility of the team, while maximizing the flexibility of the team usually requires making shared, redundant, and hence ambiguous role assignments. To balance these two conflicting goals, we need a practical and flexible computational framework for representing and reasoning about the overlapping shared mental models among teammates. Such a framework would enable an agent to dynamically reason about the status of the entire team and adapt its behavior accordingly.

Representing a shared mental model is a challenging problem. It can be viewed as a kind of belief reasoning, which is generally intractable [Halpern and Moses, 1992]. Moreover, the content of the shared mental model is quite broad, ranging from shared domain knowledge to a common relevant picture of the situation. We tackle this problem by focusing on two specific uses of the shared mental model: making teamwork efficient through *anticipating* the actions and expectations of others (e.g. by knowing others' roles, capabilities, and commitments), and by *information exchange* (knowing who to ask for information, or providing information proactively just when it is needed by someone else to accomplish their task). To avoid issues of computational complexity with belief reasoning (e.g. higher-order modal logics), we use Petri Nets as an approximate finite and computable model of mental states.

The Petri Net is a natural representation for parallel action and synchronization in a multi-agent world [Sowa, 2000]. Transitions can represent actions, with input places corresponding to pre-conditions and output places corresponding to effects. We extend the standard (colored) Petri Net formalism with special kinds of places called *control nodes* and *belief nodes*. Control nodes represent the belief an agent has about the current goals and activities of others in the team. Belief nodes represent the belief an agent has about the world, when coupled with a unification-based theorem-prover, can represent first-order knowledge, including dynamic facts and inferences about the world. In addition to serving as the shared mental model, Petri Nets also play the dual role of monitoring and tracking the execution of team plans.

CAST generates the Petri Net-based representation using two kinds of knowledge: 1) team structures (roles and responsibilities), and 2) teamwork process knowledge (e.g., individual plans, team plans). They are described in a knowledge representation language called MALLET (a Multi-Agent Logic-based Language for Encoding Teamwork). A Petri Net for a given team member represents both the background knowledge for their individual responsibilities (e.g. goals, operators), and how their role is integrated with the rest of the team.

Based on the shared mental model, the CAST kernel enables CAST agents to decide on the fly how to accomplish desired goals, how to select responsibilities to commit to or delegate, how to proactively assist others in the team, and how to effectively communicate within the team. This is achieved by dynamic role selection and proactive information change. We will give a more detailed description in later sections.

3 Formal Foundation of Proactive Information Flows

In this section, we discuss the formal foundations underlying the type of information exchange addressed in CAST. In particular, we think it is important to specify, *under ideal conditions*, what information should be exchanged between whom, and at what time. The purpose of information exchange must be oriented toward improving the efficiency or performance of a team, but otherwise is desired to be kept to a minimum to avoid the cost of communications overhead (however this is defined in the domain). In fact, a quantitative utility function that incorporates such costs is actually used in STEAM [Tambe, 1997] to help evaluate tradeoffs and decide when to communicate within a team.

In CAST, we take a different approach to optimizing information exchange by defining narrow criteria for the exchange of only the most critical information. Specifically, we want agents who know some fact to communicate it to exactly those teammates who need the information in the present context to carry out their goals,

and who furthermore (probably) do not already know it. Clearly, this might involve some complex reasoning about beliefs and goals, as well as tracking the state of other agents. However, to the degree that some of these inferences can be approximated, the team members can make intelligent decisions to selectively choose their interactions with one another.

We start by defining a simple belief language and model theory to be able to talk about the mental states of various. We model beliefs using a modal operator BEL, e.g. `(BEL bill (have joe hammer))`, with the usual possible worlds semantics [Cohen & Levesque, 1990]. We need to be able to talk about 'pieces' of information, which in this present context refer *syntactically* to sentences, but *semantically* are equivalent to constraints over possible worlds, i.e. those worlds satisfying the expression. Goals, however, refer to specific steps in plans in MALLET, to which agents can make commitments.

Using this framework, we can formally characterize the (normative) conditions under which information exchange should take place. Information *I* should be sent from one agent *A* to another agent *B* when: 1) agent *A* knows the truth-value of *I*, 2) agent *A* believes that agent *B* does not currently know *I*, and 3) *B* has a current goal *G*, the achievement of which depends on knowing *I*, i.e. if *B* does not believe *I*, then it will never be able to accomplish it's goal, but if it knew *I*, it would be able to:

$$(\text{BEL A I}) \wedge (\text{BEL A } \neg(\text{BEL B I})) \wedge$$
$$(\text{BEL A (GOAL B G)}) \wedge$$
$$[\neg(\text{BEL B I}) \rightarrow \Box\neg(\text{DONE B G})] \wedge$$
$$[(\text{BEL B I}) \rightarrow \neg\Box\neg(\text{DONE B G})]$$
$$\rightarrow (\text{GOAL A (Inform B I)})$$

where $\Box$ is the temporal operator for 'always'. In CAST, we use the pre-conditions of operators to determine the information *I* that an agent needs to know to achieve its goals. Note, the communication is suppressed only when (*A* believes that) *B* already believes *I* is true; but if *B* does not have a belief about the truth value of *I* at all, or *B* *incorrectly* believes it is false, then the message will be sent. For simplicity, we assume that all agents share common (and correct) knowledge of the team, e.g. team goals and plans, roles and responsibilities, operator pre-conditions, etc.

Clearly this approach requires *A* to monitor *B*'s mental state, both in terms of what his beliefs and goals (commitments) are. This might be easy to do in a highly observable environment, but might require extensive communication in other cases (e.g. verbal updates of what each teammate is currently doing). Alternatively, agents might use a probabilistic mechanism like Bayesian reasoning to infer the likely states of their teammates, based on observations of the effects of their actions reflected in the environment. Regardless of how difficult this state estimation or tracking might be to implement, the definition above describes the *ideal* conditions under which one would want to communicate. As much as possible, we want to restrict communication to cases where it can be inferred to be useful, which is what the DIARG algorithm below is designed to approximate.

4 Specifying Team Knowledge in MALLET

In this section, we briefly describe MALLET, the team knowledge representation language in CAST. MALLET provides descriptors for encoding knowledge about operators and plans of individuals and the team, as well as definitions of roles and responsibilities on the team. An important ontological commitment in MALLET is that *roles* are fundamentally treated as *references to specific steps in team plans*, to which certain agents on the team can externally be assigned. That is, the meaning of a role is defined by a certain step in a team plan that is dedicated for the agents playing that role to carry out.

MALLET syntax is based loosely on LISP, in the sense of using s-expressions and prefix notation. Variables are indicated with '?' prefix. Operators are simply names of atomic actions that can be taken in the environment. Operators may list a set of pre-conditions and/or post-conditions, each as a conjunction of literals (first-order predicates or their negations). Here is an example for climbing over a pit:

```
(operator climb-over (?pit)
(pre-cond (have-ladder)))
```

There are two types of operators: *individual* operators and *team* operators. Individual operators are assumed to be executed by only one agent at a time. However, team operators have the possibility of being invoked on a *set* of agents (e.g. those playing a given role). How the agents handle the team operators depends on what sub-type it has. We have identified three modes of operator-sharing:

- AND operators, which require simultaneous action by all the agents involved
- XOR operators, which require at most one agent to act (mutual exclusion, e.g. to avoid conflicts)
- OR operators, which can be execute by any of the agents (possibly >1) without conflict

Hence team operators contain an extra component that defines the "share-type." For example, an operator for lifting a heavy object might be written as:

```
(t-operator lift-heavy (?x)
   (share-type AND)
   (pre-cond (movable ?x)))
   (effect (holding ?x)))
```

Plans in MALLET are essentially designed to describe *processes*. Processes consist of invocations of atomic actions, or arbitrary combinations using various constructs such as sequential, parallel, contingent, or iterative. The syntax of processes can be defined recursively according to five constructs, with obvious intuitive meanings:

```
(seq P Q), (par P Q), (if (cond C) P
[Q]), (while (cond C) P), (do T)
```

where P and Q are sub-processes, C is condition (conjunction), and T is an operator or plan instantiation (with arguments). These constructs for describing complex processes can be given a semantics in various formalisms such as dynamic logic [Harel, 1984].

Individual plans have pre-conditions and effects, and also a process description. Team plans are similar to individual plans, but they have two extra features related to assignment of roles. First, we have declarations of role variables of the form `(role <role-name> <role-variable> <constraints>)`, where the constraints are a conjunctive set of conditions that put restriction on the role variable, which candidates must satisfy. For example, the team plan below requires two agents, one with the carrier role, the other with the fighter role. The fighter agent has an additional constraint, i.e. satisfying the predicate *closestToWumpus*.

```
(t-plan explore-cave ()
 (role carrier ?ca)
 (role fighter ?fi ((closestToWumpus ?fi) )
 (process (seq
          (do ?ca (find-wumpus))
          (do ?fi (moveto-wumpus))
          (do ?fi (kill-wumpus)) ) ) )
```

The general idea is that an appropriate agent meeting the constraints will be selected from the set of those assigned to play the role, and bound to the role variable within the plan. These role variables can then be used within processes to specify which agents will do certain steps (operators or sub-plans). An important implication of allowing constraints in the role specification is that it introduces flexibility to the teamwork process because the selection of such roles needs to be made dynamically at run time to assure that the constraint is satisfied. We will elaborate on the role selection scheme in the next section.

Finally, MALLET also provides simple descriptors for defining the members on the team, the roles they play, and their capabilities and responsibilities. While capabilities are treated simplistically (as static associations between agents and operators), the meaning of a responsibility is more interesting. It is similar to a role, in the sense that it defines agents who are supposed to do certain actions, but they are not tied to specific steps in the context of specific plans. Instead, the responsibility of an agent for an operator means that, whenever the action needs to be done at any time, the agent knows that it should act or at least coordinate with others who share the responsibility.

MALLET descriptions are converted into Petri Nets on which the agent's reasoning algorithm operate. A Petri-Net generation algorithm constructs transitions for each atomic operator in a team plan, connects them with control nodes, and links their inputs and outputs to appropriate belief nodes based on pre- and post-conditions. Since plans can be hierarchical, the sub-plans are expanded by calling the Petri-Net generation algorithm recursively, and linked into the main Petri Net through control nodes such that all the action nodes ground out in operators. Thus for each team goal, there is a single Petri Net that describes its plan for solving it. By placing tokens on the topologically-first node(s) of the Petri Net, and moving them forward whenever steps are completed, agents can keep track of the progress of the team. Though not every agent is involved in or responsible for every step, this Petri-Net model of the overall team plan forms a common understanding of the team's goals and process, which they use to determine how their individual actions fit together, and is thus a primary constituent of their *shared mental model*.

The knowledge compiler also performs a static information-flow analysis [Yin et al., 2000] for the online DIARG algorithm. An information-flow relation *I* is defined as a 3-tuple, <*Info,needers,providers*>, where *Info* is the predicate name together with 0 or more arguments, *needers* is a list of agents who might need to know such information, and *providers* is a list of agents who might know the information. We determine the needers of information by analyzing the pre-conditions of operators for which each agent might be responsible. And we determine the potential providers of information by analyzing the post-conditions of operators for which the agents might be responsible. In particular, if *P* is a post-condition of operator *O*, then we assume that an agent will know *P* after executing *O*, since we can expect agents to know the (direct) consequences of their own actions.

5 CAST Agent Kernel

The CAST kernel refers to a set of algorithms that CAST agents use to determine what actions and communications they will take at each time step. CAST agents execute a standard *sense/decide/act* loop. During the *sense* phase, they make queries to the simulation server to update their knowledge of the state of the world. Agents also check a queue for messages from other agents at this time. During the *decide* phase, each agent examines the Petri-Net representation of the team plan to see if there are any pending actions for which it is responsible. In cases of ambiguity, the agents might have to communicate in order to determine who will take the action and when. Before finally taking actions, the agents attempt to determine if there are any interactions they can initiate. For proactive information exchange, this is accomplished by DIARG.

CAST also uses a back-chaining theorem-prover called JARE (also implemented in Java) for making inferences using domain knowledge written in the form of a separate Horn-clause knowledge base. JARE is used to determine the truth-value of conditions or constraints that need to be evaluated in interpreting MALLET expressions at run-time.

5.1 Dynamic Role Selection

The algorithm for Dynamic Role Selection (DRS) is used to support reasoning about role assignments and to generate

the necessary interactions among agents to correctly execute the team plan with appropriate actors for each step. It has the following main steps:

```
DRS(step)
1   let A be the set of agents potentially
    involved in the step, A=InvolvedAgents(step)
2   remove from A any agents that are incapable of
    taking the action
3   remove from A any agents that do not satisfy
    the role constraints, if defined for that step
4   if A is empty, do nothing
5   if |A|=1 and member(self,A), do(step)
6   else
    6a. if step is an AND operator,
           synchronize(step)
    6b. if step is an OR operator,
           attempt-in-parallel(step)
    6c. if step is an XOR operator,
           disambiguate(step)
```

For each active step in the team plan (nodes marked with a token in the process net), DRS starts by determining the set of agents that could be involved. This is done by considering several different ways of assignment, in a particular order of precedence:

```
InvolvedAgents(step)
   if step is of the form do(agent,action),
        then return {agent}
   if step is of the form do(role,action),
        then return {Ai} forall agents that were
        assigned to play that role,
        Ai∈RoleSet(role)
    if step refers to an operator op for which
        some agent has been assigned
        responsibility, then return {Ai} forall
        agents such that Resp(Ai,op)
   else return all the agents on the team,
        {Ai} forall agents such that Ai∈team
```

Assignments of specific agents are considered first, and then role specifications, where several agents may be assigned to the role for a given step in the plan. If no agents or roles are directly assigned, then the search for involved agents expands to considering any agents that might be responsible in general for such actions. Finally, if it is still undefined who should perform a step, then all the agents in the team are included, since this is ultimately a step in a plan to which they are all jointly committed.

After determining the set of agents potentially involved in the action, those agents that are unacceptable are filtered out. For example, if an agent is incapable of performing the action, then it is removed from the set. In addition, if any constraints were associated with the role definition, then only those agents that satisfy the constraints are retained. This is accomplished by making a query to the agent's knowledge base that is constructed from the conjunctive conditions of the constraint by substituting any variables that are bound in the current scope, plus replacing the agent/role variable with the identity of each candidate. Those agents that do not satisfy the query are removed from the set of involved agents. Other factors could also be considered at this stage, such as removing agents from consideration whose workload is too high.

After determining the set of involved agents, steps 4-6 are used by the agents to decide what to do based on the size of this set. If the set is empty, then there is simply nothing to do; the team must wait until an acceptable agent becomes available. If there is a single (unique) agent involved in the step, then the agent may act right away, without any coordination. In these cases, CAST agents are intelligent enough to act on their own. This illustrates one of the sources of efficiency in teamwork gained from *a priori* role assignments - agents can sometimes figure out what to do on their own. If there are multiple agents involved in the step (line 6 in the algorithm), then each agent's response depends on the type of operator it is. If it is an AND operator, then all the involved agents must act simultaneously. This synchronization can be accomplished by broadcasting a *ready* signal and waiting until they have heard the same from all the other agents. If the operator is an OR operator, then any of the agents may take the action. Hence they all commit to trying to carry it out. Several might actually succeed (independently) in performing the act. However, not be too wasteful, we require agents to broadcast a *success* message when they have completed the step, at which point the other agents involved are permitted to drop their commitment (this is what `attempt-in-parallel` means in step 6b of the DRS algorithm above). Finally, if the step refers to an XOR operator, only one of the agents involved may take the action, otherwise some interference might occur. Therefore, agents must agree amongst themselves who will take responsibility for the action. This disambiguation can be accomplished through a variety of protocols, such as first-come-first-served, but they all require communication. Hence agents exchange messages to select a delegate, and then this agent make a commitment to do the action and the others do not have to.

While the current algorithm for Dynamic Role Selection produces flexible teamwork that is adaptive to specific situations, it does not handle all possible situations. For example, if agents could die, lose their connection to the team, or become incapable dynamically within the environment, then the team might need to react to these changes and adjust its operation. The possibilities range on a spectrum of complexity from using redundant role assignments for providing backup behavior and load-balancing, to dynamic re-configuration of the whole team. These features could be added to the DRS algorithm, but are left for future work.

5.2 DIARG algorithm

The DIARG algorithm (Dynamic Inter-Agent Rule Generator), an extension of IARG [Yin et al., 2000]), is responsible for identifying opportunities for proactive information exchange. Instead of sending information to all

possible needers and asking all possible providers for information, agents with DIARG can keep track of the teamwork status and only send information to whom it is relevant in the current context, and ask those agent who might know the information at the moment. This makes the DIARG algorithm more dynamic. DIARG is run by each agent during each cycle, independent of the other decision-making activities, and could result in agents sending additional messages to one another. In particular, agents attempt to identify pieces of information that other agents might need to know and inform them via TELL operations. In addition, if agents need some information and they can figure out who might know it, they can generate ASK operations. These inferences are accomplished partly by examining the information-flow relations, coupled with analysis of the current state of the team.

```
ProactiveTell()
  If I is a newly-sensed piece of information,
      or I is a post-condition of the last action
      P taken by self, then
    if I is not in the knowledge base,
      assert I into knowledge base
    for each information-flow
        <predicate, needers, providers>
      if I matches predicate name and self is
          included in the providers, then
        for each agent x in the needers,
            if agent x plays a role in an active
            step,
            TELL(x,I)

ActiveAsk()
   for each active step s in the plan in which
   self is involved
      let o be the operator to which s refers
      for each pre-condition I of s
        if not(know(self, I)), then
          let Info-flow = <predicate, needers,
          providers> be the information flow in
          which I matches predicate name
          select the agent y from providers
          (active agents first),
          do ASK(y,I)
```

These algorithms are designed to generate inter-agent communications based on approximation of the criteria set forth in Section 3 for ideal conditions under which it is desirable to exchange information. In principle, we want agents to communicate only when it is likely to be useful, to minimize network traffic (use of bandwidth). The formal criteria are difficult to achieve in practice, particularly due to the need for belief reasoning. While we do not model the complete belief state of other agents, our static analysis of information flow identifies agents which might know or need to know certain information based on their involvement in the team plan (i.e. through actions for which they might be responsible). When some information changes, for example due to the action of an agent, and there is an effect of the action that others cannot observe, or when an agent observes some new information, then it might want to inform the others (proactively). The sending of these messages is restricted to those agents who conceivably might need to know the information. Whether or not an agent *really* needs to know a piece of information also depends on whether it supports one of their current goals, which we predict from active nodes in the Petri Net.

However, these algorithms cannot guarantee perfectly optimal information flow. For example, it depends on what they can observe and what they can infer. Also, while we assume that agents are never forgetful, information might change dynamically (non-deterministically) at different rates in different environments, and not all agents might be able to observe changes in all information, making the problem of maintaining consistent and correct distributed knowledge in a team very difficult in general [Singhal and Zyda, 1999]. However, the DIARG algorithm is implemented at an appropriate level for supporting interactions in mixed human-agent teams, since it does not require direct access to (or simulation of) the mental state of other team members, and is able to derive potentially useful information flows based only on analysis of common (shared) knowledge of the team plan, individuals' roles, and the current state of progress, which can be assumed to be monitored by all the members of the team.

6 A Testbed Application

We have constructed a testbed application based on a multi-agent extension of the Wumpus World [Russell and Norvig, 1995]. These agents can act as part of a team by playing the role of a fighter, to shoot the wumpus, or a carrier, to carry the gold they find. To generate the need for information flow between team members, the two types of roles have different sensing capabilities. While both of them can sense a stench (from a wumpus), a breeze (from pits), and glitter (from gold), only the carrier can pin-point the exact location of the wumpus when it is in an adjacent room in the cave. These experiments use a team plan and individual plans, along with other related teamwork knowledge, to find and kill multiple wumpuses and collect gold. Agents can communicate in three ways: (1) proactive tell, (2) broadcast information, and (3) broadcast control tokens for coordination purposes.

Table 1. Different teams used in experiments

Experiment	Team	#of Carriers	#of Fighters	Team work	Communication	DRS
1	A	1	1	Yes	Proactive Info Exchange	No
	B	1	1	Yes	Broadcast Every New Info	No
	C	1	1	No	No	No
2	D	1	3	Yes	Proactive Info Exchange	No
	E	1	3	Yes	Proactive Info Exchange	Yes

In order to evaluate the effectiveness of different features supported by CAST - shared mental models, proactive information exchange, and dynamic role selection - we have devised two sets of experiments and five multi-agent teams for comparison. The differences among the five teams are listed in Table 1. Experiment 1 is run on Team A, Team B, and Team C to show the benefit of using shared mental models and proactive information exchange. Experiment 2 is run on Team D and Team E to show the benefit of dynamic role selection.

In Experiment 1, Teams A and B both use teamwork, whereas C does not. Furthermore, Team A uses proactive information exchange (the DIARG algorithm is turned on), whereas agents in Team B just broadcast each new piece of information indiscriminately to all the members on the team. Agents in Teams A and B use a MALLET specification of a team plan that requires the carrier to first find the wumpus, then navigate a fighter to a room adjacent to the wumpus and shoot the wumpus. Agents in Team C merely wander around randomly, independently looking for wumpuses to shoot and gold to pick up. To make the comparison fair, agents in Team A and Team B also use individual plans in MALLET to perform this wandering behavior when not busy. So there is teamwork based on the shared mental model of the team plan for agents in Team A and Team B, but in Team C, there is no communication and no teamwork. Experiment 2 is meant to show the benefit of dynamic role selection. Team D is similar to team A except that Team D has more fighters; however, Team D does not use dynamic role selection. Team E implements the dynamic role selection protocol for each of its agents. All five teams use the same knowledge base (i.e. JARE rules) for reasoning about the environment and for determining the priority of actions.

Table 2. Comparison of performance for teams that differ in teamwork and information exchange.

Team	Static	V1	V2	V3	V4	V5
A	Average	3.6	7.35	2.6	1	2.2
	Std. Dev.	0.82	3.79	1.35	0	0.62
B	Average	3.5	6.55	2.55	138	0
	Std. Dev.	0.76	3.52	1.32	12.7	0
C	Average	1.6	7.85	2.2	0	0
	Std. Dev.	0.88	5.58	1.94	0	0

V1: # Wumpuses Killed V2: # of Arrows Used
V3: # of Gold Found V4: # of Messages Broadcast
V5: # of Proactive Information Exchanges

Experiment 1 was performed on 20 randomly generated maps for a world (cave) with 10 by 10 cells (rooms), 5 wumpuses, 2 pits, and 10 piles of gold. Each team is allowed to operate for a fixed number of total actions. The performance of each team for each case is measured by (1) the number of wumpuses killed, (2) the number of arrows used, (3) the amount of gold gathered (4) the number of messages broadcast, and (5) the number of proactive tell messages. The average and standard deviation of the experiments are summarized in Table 2.

As shown in the Table 2, Team A and Team B achieve comparable performance, because they use an identical shared mental model specified in MALLET. However, Team B has a much higher communication overhead (V4). Team A and Team B both have better performance in terms of number of wumpuses killed, because there is teamwork among agents in Team A and Team B. Whenever the carrier in Team A and Team B finds any wumpus, it will tell the fighter to go kill it, which gives the whole team a better chance to kill more wumpuses. However, picking up gold is an individual activity, so from the table, we can not see much difference among the three teams. One more advantage of Team A is that the fighter only needs one arrow to shoot a wumpus with known location. This is achieved because the carrier agent tells the fighter agent proactively about the location of the wumpus, using DIARG to infer that the fighter needs the information to navigate to the wumpus. In contrast, the fighter in Team C has to randomly choose a direction for shooting when it senses a stench, and cannot get any help from the carrier to locate the wumpus. Consequently, Team C consumes the same number of arrows to get half as many wumpuses as both Team A and Team B, and thus has worse resource utilization.

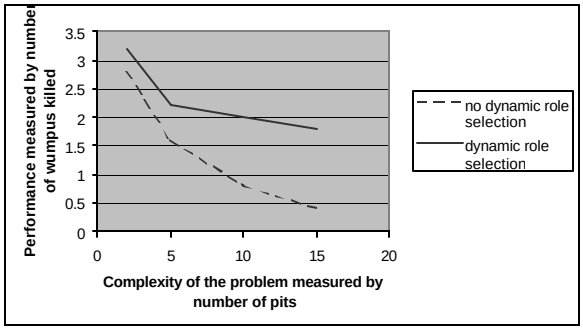


Figure 1 - The Comparison of teamwork with and without dynamic role selection in increasingly complex environments

Experiment 2 was performed on 4 sets of 5 randomly generated maps for a world with 10 by 10 cells, 5 wumpus, and 10 piles of gold. The difference between the sets of maps is that we incrementally increase the complexity of the world by changing the number of pits in the cave, from 2 to 5 to 10 to 15 pits respectively. Each team is allowed to operate for a fixed number of actions. The performance of each team for each case is measured by the same metrics we used in the first experiment. The performance difference is shown in Figure 1.

From Figure 1, we can see that with dynamic role selection, Team E performed better than Team D and the difference becomes larger when the complexity of the

problem scales up. Because the most appropriate fighter will always be chosen to kill the wumpus whenever the carrier finds one, more wumpuses can be killed in a limited number of actions. And in worlds with more pits, it is even harder for the fighter to navigate to the wumpus found, so choosing the closest fighter becomes much more important. But when the world has so many pits that even the carrier gets trapped, teamwork becomes impossible and the performance of both teams will drop significantly.

7 Conclusion

Developing a computational framework for capturing the shared mental model among members of effective teams is a challenging and critical issue for applications ranging from team training to supporting teamwork. The CAST architecture supports flexibility in its teamwork knowledge specification and in dynamic role selection at run time. At the same time, it leverages shared knowledge about the structure and the process of a team to reason about information needs of teammates efficiently. We believe the CAST architecture achieves a reasonable tradeoff between the flexibility and the efficiency for simulating proactive information exchange among teammates. While our experimental results demonstrate anticipated benefits of CAST, it also reveals some limitations of the current CAST implementation. For example, we plan to extend the role selection method for finding backups when an agent dies or becomes non-functional. With such an extension, we hope to simulate more complex teamwork behavior.

8 Acknowledgements

This research described in this paper is supported by a DOD MURI grant F49620-00-1-0326 administered through AFOSR and partially supported by internal seed funds from the College of Engineering through the Training Systems Science and Technology Initiative.

References

[Cannon-Bowers and Salas, 1997] Canon-Bowers, J. A. and Salas, E. A framework for developing team performance measures in training, in Brannick, M. T., Salas, E. and Prince, C.,editors, *Team Performance Assessment and Measurement: Theory, Methods, and Applications*, Lawrence Erlbaum Associates: Hillsdale, NJ. 1997.

[Cohen and Levesque, 1990] Cohen, P.R. and Levesque, H.J. Intention is choice with commitment. *Artificial Intelligence*, 42(3), 1990.

[Cohen and Levesque, 1991] Cohen, P.R. and Levesque, H.J. Teamwork. *Nous*, 25(4):487-512, 1991.

[Grosz and Kraus, 1996] Grosz, B., and Kraus, S. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269-357, 1996.

[Halpern and Moses, 1992] Halpern, J.Y. and Moses, Y. A guide to completeness and complexity for modal logics of knowledge and belief, *Artificial Intelligence*, 54:319-379, 1992.

[Harel, 1984] Harel, D. Dynamic logic. In *Handbook of Philosophical Logic*. Gabbay, D. & Guenther ,F. (eds.). Reidel Publishing : Dordrecht, Netherlands, 1984.

[Ilgen et al., 1993] Ilgen, D. R., Major, D. A., and Hollenbeck J. R. *Leadership and research: Perspectives and Directions*, San Diego: Academic Press, 1993.

[Jennings, 1995] Jennings, N. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. Artificial Intelligence, 75(2):195-240, 1995.

[Jones et al, 1999] Jones, R., Laird, J., Nielson, P., Coulter, K., Kenny, P., and Koss, F. Automated intelligent pilots for combat flight simulation. *AI Magazine*, 20(1):27-41, 1999.

[Rao and Georgeff, 1991] Rao, A.S. and Georgeff, M.P. Modeling rational agents within a BDI Architecture. *Principles of Knowledge Representation and Reasoning, Proceedings of the Second International Conference*, 473-484, 1991.

[Russell and Norvig, 1995] Russell, S. and Norvig P. Artificial Intelligence: *A Modern Approach. Prentice Hall*, New Jersey, 1995.

[Singhal and Zyda 1999] Singhal, S. and Zyda M. *Networked Virtual Environments*. New York, New York, ACM Press, 1999.

[Sowa, 2000] Sowa, J.F. *Knowledge Representation: Logical, Philosophical, and Computational Foundation*. Brooks/Cole: Pacific Grove, CA, 2000.

[Stone and Veloso, 1999] Stone, P., and Veloso, M. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110:241-273, 1999.

[Tambe, 1997] Tambe, M. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83-124, 1997.

[Tidhar et al., 1998] Tidhar, G. and Heinze, C. and Selvestrel, M. Flying together: Modelling air mission teams. Journal of Applied Intelligence, 1(1) pp1-1, 1998.

[Wooldridge and Jennings, 1995] Wooldridge, M., and Jennings, N., Intelligence agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115-152, 1995.

[Yin *et al*., 2000]Yin, J., Miller, M., Ioerger, T.R., Yen, J., and Volz, R.A. A knowledge-based approach for designing intelligent team training systems. In: *Proc. of the Fourth International Conference on Autonomous Agents*, pp. 427-434, 2000.

MULTI-AGENT SYSTEMS

MARKET MECHANISMS

Market Clearability*

Tuomas Sandholm
sandholm@cs.cmu.edu
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Subhash Suri
suri@cs.ucsb.edu
Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

Market mechanisms play a central role in AI as a coordination tool in multiagent systems and as an application area for algorithm design. Mechanisms where buyers are directly cleared with sellers, and thus do not require an external liquidity provider, are highly desirable for electronic marketplaces for several reasons. In this paper we study the inherent complexity of, and design algorithms for, clearing auctions and reverse auctions with multiple indistinguishable units for sale. We consider settings where bidders express their preferences via price-quantity *curves*, and settings where the bids are price-quantity *pairs*. We show that markets with piecewise linear supply/demand curves and *non-discriminatory pricing* can always be cleared in polynomial time. Surprisingly, if *discriminatory pricing* is used to clear the market, the problem becomes $\mathcal{NP}$-Complete (even for step function curves). If the price-quantity curves are all *linear*, then, in most variants, the problem admits a poly-time solution even for discriminatory pricing. When bidders express their preferences with price-quantity pairs, the problem is $\mathcal{NP}$-Complete, but solvable in pseudo-polynomial time. With free disposal, the problem admits a poly-time approximation scheme, but no such approximation scheme is possible without free disposal. We also present pseudo-polynomial algorithms for XOR bids and OR-of-XORs bids, and analyze the approximability.

1 Introduction

Market mechanisms play a central role in AI for several reasons. First, they provide a tool for resource and task allocation in multiagent systems where the agents may be self-interested. Second, AI techniques can be used to clear markets. For example, there has been a recent surge of research in the AI community on search algorithms [Sandholm, 1999; Fujishima *et al.*, 1999; Sandholm and Suri, 2000] and special-case polynomial algorithms [Tennenholtz, 2000] for clearing combinatorial auctions. Third, recent electronic commerce server prototypes such as eMediator [Sandholm, 2000] and AuctionBot [Wurman *et al.*, 1998] from academic AI groups have led to the uncovering of a need for fast clearing algorithms for a vast space of market designs.

In this paper we analyze the inherent complexity of, and design algorithms for, clearing auctions and reverse auctions in the ubiquitous setting where there are multiple indistinguishable units of an item for sale.

In the largest securities markets where there are multiple units of each item (e.g., stock) for sale, there usually are liquidity providers (*market makers* on the NASDAQ and a *specialist* on the NYSE) that carry inventory, and guarantee that trades are possible at essentially any quantity. Therefore, direct matching between buyers and sellers is not absolutely necessary. However, we argue that if technically possible, it would be highly desirable to construct market clearing algorithms that directly match buyers and sellers, and do not rely on an external liquidity provider, for several reasons. First, most ecommerce marketplaces do not have external liquidity providers. Second, liquidity providers incur operating cost, and need to be compensated. This compensation tends to be paid (implicitly) by the market participants. Third, if the items that are being traded are not securities, the SEC does not impose or monitor rules on liquidity-provisioning parties. Finally, even in markets where the liquidity providers are regulated, they frequently violate the regulations (the most famous recent cases are from the NASDAQ).

We study the possibility of algorithms for accomplishing this in the context of price-quantity curves first, and price-quantity pairs second. We analyze markets with and without free disposal of units. We also uncover the complexity implications of non-discriminatory vs. discriminatory pricing.

2 Auctions with Demand Curve Bids

We consider the auction setting where each bidder submits a *demand curve* indicating the quantity $q(p)$ he will accept at each *unit price* p. If his bid is cleared at price p, he receives $q(p)$ units, for a total price of $p \cdot q(p)$. Recently, several computational markets have been built that use piecewise linear [Sandholm, 2000] or step function [Lupien and Rickard, 1997; Sandholm, 2000] demand curves.

We focus mainly on piecewise linear curves because they can approximate any curve arbitrarily closely, and because their complexity (in the sense of measuring the length of the input) can be characterized systematically. In order to keep our discussion simple, we use a single parameter k to denote the complexity of the piecewise linear curves. That is, k is the *largest* number of pieces in any bidder's demand curve.

We begin with an elementary lemma, which will be used repeatedly in the following discussion. For now, in order to keep the discussion simple, let us assume that the seller has an infinite supply of units to sell. When the number of units available is finite, the solution follows as an easy corollary of Lemmata 2.1 and 2.2, as is discussed after those lemmata.

Lemma 2.1 *Consider a linear demand curve* $q = ap + b$ *subject to* $p, q \geq 0$.

*This work was funded by, and conducted at, CombineNet, Inc., 311 S. Craig St., Pittsburgh, PA 15213.

- *If $a \geq 0$, then an infinite revenue is achievable.*
- *If $a < 0$, then the revenue is maximized at price $p^* = -\frac{b}{2a}$. The corresponding quantity sold at this price is $q^* = \frac{b}{2}$, and the revenue is $-\frac{b^2}{4a}$.*

PROOF. If the slope is non-negative, the revenue is maximized by setting $p^* = \infty$. The second case is more interesting. The revenue at price p equals $p(ap + b) = ap^2 + bp$. Setting the first derivative with respect to p to zero, we get $2ap = -b$, which yields $p^* = -\frac{b}{2a}$. The second derivative is negative (since $a < 0$), so the revenue is maximized at this p. The values of q and $p \cdot q$ follow easily. □

The demand curve could also have boundaries, meaning that the bidder expresses his preference by specifying the linear demand curve $q = ap + b$ but with explicit bounds on the price. If the curve is restricted to the price range $[p_1, p_2]$, where $p_1 < p_2$, we call it a *bounded linear demand curve.*

Lemma 2.2 *Consider a bounded linear demand curve, $q = ap + b$, restricted to the price range $[p_1, p_2]$.*

- *If $a \geq 0$, then the revenue is maximized at price $p^* = p_2$.*
- *If $a < 0$, then the revenue is maximized either at $p^* = -\frac{b}{2a}$ provided $p_1 \leq -\frac{b}{2a} \leq p_2$, or at that endpoint of the range $[p_1, p_2]$ which is closer to $-\frac{b}{2a}$.*

PROOF. Let us first consider the case of non-negative slope, $a \geq 0$. When $a \geq 0$, we have $q' > q''$ whenever $p' > p''$, and so $p'q' > p''q''$. Thus, the maximum revenue is achieved at the highest permissible price, which is p_2.

When the demand curve has negative slope, Lemma 2.1 tells us that the optimal solution without price boundaries is $p^* = -b/2a$. If this optimal price is within the range $[p_1, p_2]$, then it obviously maximizes the revenue, and we are done. So, let us now assume that $p^* \notin [p_1, p_2]$.

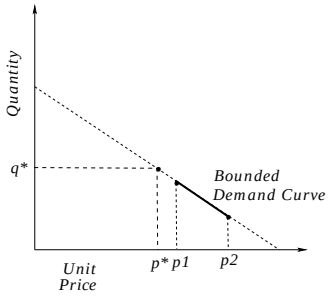


Figure 1: Revenue maximization for a linear demand curve with boundaries.

We consider the effect on revenue of changing the price by an amount ε from the unconstrained optimum $p^* = -b/2a$. (See Figure 1.) Let (p', q'), where $q' = q(p)$, be an arbitrary point on the linear curve $q = ap+b$. Since a is the slope of the demand curve, we note that $\frac{q'-q^*}{p'-p^*} = a$. Let $\varepsilon = p' - p^*$ be the change in the price, and let $\delta = q' - q^*$ be the corresponding change in the quantity. Then, it follows that $\delta = a\varepsilon$. The revenue at point (p', q') is

$$\begin{aligned} p'q' &= (p^* + \varepsilon) \cdot (q^* + \delta) \\ &= p^*q^* + \varepsilon q^* + \delta p^* + \varepsilon\delta \\ &= p^*q^* + \varepsilon q^* + a\varepsilon p^* + a\varepsilon^2 \\ &= p^*q^* + \varepsilon(b/2) + a\varepsilon(-b/2a) + a\varepsilon^2 \\ &= p^*q^* + a\varepsilon^2 \end{aligned}$$

In line 4 we used the fact that $p^* = -\frac{b}{2a}$ and $q^* = \frac{b}{2}$. Since the demand curve slope a is negative, this shows that a change of ε from the unconstrained optimal price $p^* = -\frac{b}{2a}$ *reduces* the revenue by $|a|\varepsilon^2$. Thus, if the price curve is bound to the range $[p_1, p_2]$, and $p^* \notin [p_1, p_2]$, the maximum revenue is achieved at either p_1 or p_2, whichever is *closer* to p^*. □

In the preceding discussion, we assumed that the seller has an unlimited supply of units. When this supply is bounded by some quantity Q, the optimal solution can be derived easily, as follows. Of course, a feasible solution exists if and only if $Q \geq \min\{q_1, q_2\}$. When the slope is positive, the seller sells $\min\{q_2, Q\}$ units. When the slope is negative and $Q \geq b/2$, revenue is maximized at $q^* = b/2$. But if $Q < b/2$, then the revenue is maximized at $q^* = Q$.

A *piecewise linear* demand curve consists of one or more bounded linear curves. We do not require the demand curve to be continuous (i.e., the quantity can "jump" between pieces). Given a single piecewise linear demand curve bid, we can use Lemma 2.2 on each linear piece separately to determine the revenue-maximizing allocation. To lay the groundwork for our auction-clearing algorithm, we next discuss the problem of *aggregating* the demand over multiple piecewise linear curves.

Demand Curve Aggregation: Consider a set of piecewise linear curves $f_1, f_2, \ldots, f_n$. Their *aggregate curve* is a piecewise linear function $f : R^+ \to R^+$ such that $f(p)$ is the total demand at unit price p. That is, $f(p) = f_1(p) + f_2(p) + \ldots + f_n(p)$, where $f_i(p)$ is the demand by curve i at unit price p. For instance, if the demand curves are linear functions $q = a_ip + b_i$, $i = 1, 2, \ldots, n$, then their aggregate curve is easily shown to be the linear function $q = (\sum_i a_i)p + \sum_i b_i$.

Breakpoints of Aggregate Curve: The aggregate curve f changes only when one of the component curve changes; that is, the breakpoints of f are the union of the breakpoints of the component curves. Thus, given a set of n piecewise linear curves each of which has at most k pieces, their aggregate curve f has at most nk breakpoints.

Given n piecewise linear curves, we can compute their aggregate curve in time $O(nk \log(nk))$, as follows, where k is maximum number of pieces in any curve. Let $z_1, z_2, \ldots, z_L$, where $L \leq nk$, denote the breakpoints of all the component curves, in sorted order. We scan these breakpoints in right to left order (decreasing order of price), and determine the linear aggregate curve between two consecutive breakpoints.

Initially, we compute the linear aggregate function in the range (z_L, ∞), in $O(n)$ time. Next, as we move to the next breakpoint, at most one linear piece changes—one piece may end and another may begin. (If multiple curves begin or end at the same point, we can enforce an artificial order among those, and consider them one at a time.) We can update the linear aggregate by deleting the coefficients of the leaving curve and adding those of the entering curve, and so each update takes $O(1)$ time. Thus, the complete aggregate curve can be determined in time $O(nk)$, after an initial sorting cost of $O(nk \log(nk))$.

While the construction just described should be sufficiently fast for most practical applications, we can build the aggregate curve even faster if only part of the curve is needed.

Lemma 2.3 *Given n piecewise linear demand curves, we can construct the m rightmost pieces of their aggregate curve in time $O(m \log n + n)$, where $m = O(nk)$, and each curve has at most k pieces.*

PROOF. We maintain a priority queue that stores the "next" breakpoint (end, begin, or change) of each component curve. We process events in the order presented by the priority queue: when we delete a breakpoint from the queue, we insert the next linear piece (if any) of the same curve. We initially compute the rightmost piece of the aggregate curve in $O(n)$ time. After that the aggregate curve is updated at each event point in $O(1)$ time. Inserting or deleting an event from the priority queue takes $O(\log n)$ time, and so the total cost to construct m rightmost pieces of the aggregate curve is $O(m \log n + n)$. □

2.1 Auctions with Non-Discriminatory Pricing

Say that the seller wants to auction off Q indistinguishable units of an item. Each of the n bidders submits a piecewise linear demand curve bid. In a *non-discriminatory* auction, the seller determines an optimal price p^* to maximize his revenue, and every buyer pays the same unit price p^*. (The number of items received by bidder i is computed using his demand curve, evaluated at price p^*.)

As Lemma 2.2 shows, if the seller wants to maximize his revenue, he might not sell all the units. We therefore consider the auction both with and without *free disposal*. With free disposal, the seller may choose to keep some units (because he can dispose of them for free), but without free disposal, he must sell all the units.

Theorem 2.4 *Consider a single-item, multi-unit auction with n bidders, each with a piecewise linear demand curve. Under non-discriminatory pricing, the auction can be cleared so as to maximize the seller's revenue in time $O(nk \log(nk))$ with or without free disposal, where k is the maximum number of pieces in any bidder's demand curve.*

PROOF.

- [*Without Free Disposal.*] We construct the aggregate demand curve, incrementally from the right, as described in Lemma 2.3. For each linear piece of the aggregate curve, we check to see if it intersects the supply line $q = Q$. If there is an intersection, then the intersection point is a feasible solution. Since the goal is to maximize seller's revenue, we want the rightmost (highest price) intersection. Thus, we can stop the algorithm as soon as we find an intersection. From this price we can determine the quantities sold to each bidder, using their demand curves. (The problem is clearly infeasible when there is no intersection between the line and the aggregate curve.)
- [*With Free Disposal.*] In this case, we compute the entire aggregate curve, since we cannot stop at the rightmost feasible solution. For each linear piece of the aggregate curve, we compute the maximum feasible revenue and keep track of the optimum found so far.
  1. If the piece lies entirely above the line $q = Q$, no feasible solution exists for this piece.
  2. If the piece is entirely below the line, we take the solution given by Lemma 2.2. In other words, we compute the unconstrained optimum p^* for this linear curve. If p^* is within the price bounds of the piece, we take that solution; otherwise, we choose the endpoint of the linear piece whose price is closer to p^*.
  3. If the piece intersects the line $q = Q$, we compute the unconstrained optimum q^*. If $Q > q^*$, we take the unconstrained solution; otherwise, we sell Q units.

  Each of the three cases takes constant time to evaluate. Thus the complexity is dominated by the time to build the aggregate curve, which is $O(nk \log(nk))$. □

2.2 Auctions with Discriminatory Pricing

In a *discriminatory* price auction, the seller determines for each buyer j a distinct unit price p_j with the objective of maximizing his revenue $\sum_j p_j q_j$ subject to the supply constraint $\sum_j q_j \leq Q$. The quantity q_j sold to buyer j is determined from j's demand curve at price p_j. For a fixed set of demand curve bids, the seller's revenue in a discriminatory auction is generally higher (and never lower) than in a non-discriminatory auction, but the latter offers a stronger notion of fairness among bidders. Discriminatory price auctions however do offer a weak form of *ex ante* fairness: they are anonymous in the sense that had two players swapped their bids, their allocations would also have been swapped.

Intractability under Piecewise Linear Demand Curves
In sharp contrast to a non-discriminatory auction, we show that clearing a discriminatory auction with piecewise linear demand curves is $\mathcal{NP}$-Complete. In fact, this complexity jump occurs even for the simplest piecewise linear demand curve, a *step function*.

Step Function Demand Curve: A step function demand curve is defined by a tuple (p_i, q_i), indicating a buyer's willingness to buy q_i units at or below the unit price p_i; the buyer is not willing to buy any units at price strictly greater than p_i.

Theorem 2.5 *Consider a single-item, multi-unit auction with n bidders, each making a step function demand curve bid. Then the problem of determining a revenue-maximizing allocation using discriminatory pricing, is $\mathcal{NP}$-Complete. This holds with or without free disposal.*

PROOF.

- [*With Free Disposal.*]
  We reduce the **knapsack** problem to our auction problem. Let $\{(s_1, v_1), (s_2, v_2), \ldots, (s_n, v_n),\ K\}$ be an instance of the knapsack problem—K is the knapsack capacity, s_i and v_i, respectively, are the size and value of item i. The goal is to choose a subset of items of maximum value with total size at most K. We create an instance of the single-item multi-unit auction using step function demand curves, as follows. Bidder i places a step function bid $(v_i/s_i, s_i)$, meaning he is willing to buy s_i units at lot price v_i (or maximum unit price v_i/s_i), and no units for a higher price. The total number of units available is K. Since we are using discriminatory pricing, the goal is to choose a subset of bids maximizing

the total revenue subject to the total quantity constraint K. Now, it is easy to see that any solution to the auction problem is a solution to the knapsack, and vice versa.

- [*Without Free Disposal.*]
  We reduce the **subset sum** problem [Garey and Johnson, 1979] to the auction problem. In the subset sum problem, we are given a set of integers $X = \{x_1, x_2, \ldots, x_n\}$, an integer K, and the goal is to choose a subset of X whose elements sum to exactly K. We create n bids, where the bidder i places a step function demand curve bid $(\$1, x_i)$—that is, the buyer is willing to pay one dollar per unit for x_i units, but does not accept any other quantity. (Actually, the price is immaterial in this transformation, so we use a default value of \$1.) The total number of units available is K. It is easy to see that the auction without free disposal has a feasible solution if and only if the original subset sum problem has a solution. □

Thus, we conclude that even with step function demand curve bids, or more generally piecewise linear curve bids, the discriminatory auction becomes intractable.

Polynomial Algorithm for Linear Demand Curves

There is an important case of the discriminatory price auction for which we can clear the market in polynomial time. This is the case where all bids are *downward sloping linear demand curves*, that is, in the demand curve $q = ap+b$, we have $a < 0$ and $b \geq 0$. In other words, the buyer's demand decreases linearly as the price increases. We begin by describing the algorithm for auctions with free disposal. Let us suppose that the auctioneer has Q units of the item for sale. Then, the algorithm has the following steps.

1. Let $S = \{1, 2, \ldots, n\}$ denote the index set of bids.
2. Compute the unconstrained optimal solution for each bid independently: $(p_j, q_j) = \left(\frac{-b_j}{2a_j}, \frac{b_j}{2}\right)$.
3. If $\sum_{j \in S} q_j \leq Q$, then these unconstrained price-quantity pairs are the optimal solution.
4. Otherwise, let $\delta = \min_{j \in S} \{p_j\}$, and let ℓ be the index of the bid that achieves this minimum.
5. Set $p'_j = p_j + p_\ell$, and $q'_j = a_j p'_j + b_j$, for $j \in S$. That is, increase each bid's unit price by δ, and determine the new quantity. (Note that $q'_j < q_j$ because a_j is negative.)
6. If $\sum_{j \in S} q'_j \leq Q$, then set $q^*_j = q_j - a_j C$, where $C = (\sum_{j \in S} b_j - 2Q)/(2\sum_{j \in S} a_j)$, and stop. This is the optimal quantity sold to buyer j, at the corresponding price $p^*_j = p_j - C$. (Note that both a_j and C are negative, and so $q^*_j < q_j$ and $p^*_j > p_j$.)
7. Otherwise, set $S = S - \{\ell\}$, and go to step 4.

The correctness of the algorithm depends on three facts: (a) if the unconstrained solution is quantity-infeasible, then one must increase the unit price *uniformly* for all bids, until either the solution becomes feasible or the price becomes infeasible for some bid, (b) when the price becomes infeasible for some bid, that bid receives zero units in the optimal solution, and (c) in the price-interval where the set of feasible bids remains constant, the formula of line (6) gives the optimal solution. Due to lack of space, we omit the proofs of these claims. Instead, we briefly discuss the time complexity of the algorithm. Step (4) repeatedly chooses a bid with the next smallest unconstrained optimal price p_j. In order to facilitate this choice, we initially sort the bids in increasing order of p_j's. The key step is to determine whether $\sum_{j \in S} q'_j \leq Q$. However, it is easy to see that $\sum_{j \in S} q'_j = \sum_{j \in S} q_j + (\sum_{j \in S} a_j)p_l$. Thus, we can calculate $\sum_{j \in S} q'_j$ in $O(1)$ time, by simply keeping track of the sums $\sum_{j \in S} a_j$ and $\sum_{j \in S} q_j$. These sums only change when a bid is removed from S, and can be maintained in $O(1)$ time per update to S. In summary, once the bids have been sorted, the total cost of running the algorithm is $O(n)$. We summarize this result in the following theorem.

Theorem 2.6 *Consider a single-item, multi-unit auction with n bidders, each making a downward sloping linear demand curve bid. Under discriminatory pricing, the auction can be cleared so as to maximize revenue in $O(n \log n)$ time, with or without free disposal.* [1]

Remark. For completeness, we now discuss the complexity of clearing absurd types of linear demand curves. If all the demand curves are upward sloping (the higher the unit price, the more the bidder will buy), then the optimal solution is easily obtained (with or without free disposal) by selling all Q units to the bidder whose demand line intersects the constant line $q = Q$ at the *highest* unit price. This can be determined in $O(n)$ time by calculating the intersection point for every demand line. Next we consider the case where all demand curves are constant ($a = 0$), i.e., the bidders do not care about price. With free disposal, the seller's revenue can be maximized in $O(n)$ time by choosing any bidder (whose quantity is positive, i.e., $b > 0$), and charging an infinite price. Without free disposal, finding a feasible solution (finding a combination of the constant curves whose quantities sum up to exactly Q) is $\mathcal{NP}$-Complete because that is equivalent to the *subset sum* problem. If a feasible solution is found, the seller can charge an infinite price.

3 Reverse Auctions with Supply Curve Bids

In this section, we consider reverse (or, buyer-based) auctions using an analog of demand curve bidding. These types of auctions are frequently used for Requests for Proposals (RFPs) and Requests for Quotes (RFQs). The buyer posts the items (goods, services, etc.) she wants to purchase, and sellers compete for the business by bidding to sell the items. We assume that a buyer wishes to acquire Q indistinguishable units of a certain item, and each seller submits a *supply curve bid*.

We start by establishing simple properties of *linear supply curves*. As in auctions, we consider two settings: free disposal and no free disposal. Under free disposal, the buyer is willing to accept more than the desired Q units if it leads to lower cost (at worst he can dispose of the extra units for

[1] When free disposal is not allowed, the only change to the algorithm is this: if the initial unconstrained optimal solution is quantity-feasible, meaning $\sum_{j \in S} q_j \leq Q$, instead of stopping, we *decrease* the price uniformly over all bids, until either the total demand reaches Q, or some bid becomes infeasible (its price reaches zero). When a bid becomes infeasible, we remove it from S and continue.

free). Without free disposal, the buyer wants exactly Q units (or none if the solution is infeasible).

Lemma 3.1 *Consider a linear supply curve $q = ap + b$ subject to $p, q \geq 0$. Suppose that the buyer wishes to acquire Q units of the item.*

- *If $a \leq 0$, then there is a feasible solution if and only if $Q \leq b$. If $Q \leq b$, the cost to the buyer is 0 with free disposal, and $Q(Q-b)/a$ without free disposal.*
- *If $a > 0$, then the cost of acquiring the items is $\max\{0,\ Q(Q-b)/a\}$.*

PROOF. When the supply curve has non-positive slope, the maximum number of items that can be purchased is b. Thus, the trade is feasible if and only if $Q \leq b$. If free disposal is allowed, the buyer buys b units at price 0; without free disposal, the buyer pays $(Q-b)/a$ per unit.

If the supply curve has positive slope, the price per unit is increasing with the number of units. If $b > Q$, then the buyer can buy Q units at zero cost. Otherwise, the buyer purchases exactly Q units at the unit cost of $(Q-b)/a$. □

Lemma 3.2 *Consider a bounded linear supply curve, $q = ap + b$, restricted to the price range $[p_1, p_2]$, where $p_1 < p_2$. Suppose the buyer wishes to acquire Q units of the item.*

- *Without free disposal, a feasible solution exists iff $q_1 \geq Q \geq q_2$, and if feasible, the solution has cost $Q(Q-b)/a$.*
- *With free disposal and $a \leq 0$, a feasible solution exists iff $Q \leq q_1$. Set $q_2' = \max\{q_2, Q\}$. If feasible, the solution has cost $\min\{p_1 q_1, p_2' q_2'\}$, where p_2' is the price corresponding to q_2'.*
- *With free disposal and $a > 0$, a feasible solution exists iff $Q \leq q_2$, and in that case the solution has cost $\max\{p_1 q_1, Q(Q-b)/a\}$.*

PROOF. See Figure 2 for illustration. The first case follows easily, since the boundaries of the supply curve dictate that the quantity supplied is in the range $[q_1, q_2]$. If Q lies in this range, the cost equals $Q(Q-b)/a$.

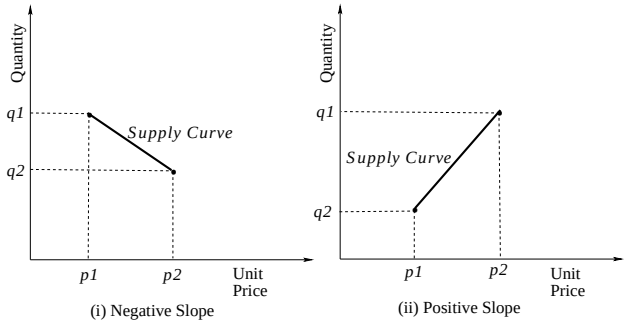


Figure 2: Bounded linear supply curves.

In the second case, since the slope is negative, the maximum quantity supplied is q_1, and so for feasibility, we must have $Q \leq q_1$. Since the buyer admits free disposal, we can purchase any quantity between Q and q_1, in order to minimize the cost. We define $q_2' = \max\{q_2, Q\}$ so that the feasible range of quantity is $[q_1, q_2']$. We now invoke the algebra of Lemma 2.2 to observe that quantity $p \cdot q(p)$ decreases the farther the price deviates from $p^* = -\frac{b}{2a}$. (Recall that we have $a \leq 0$ in this case.) Thus, the cost of acquiring at least Q units in the range $[q_1, q_2']$ is minimized at one of the endpoints of the range. So, we take the smaller of the two values.

Finally, in the third case, the largest quantity supplied by the curve is q_2, so the feasibility condition checks whether $Q \leq q_2$. If $Q \leq q_2$, then we can set $q_1' = \max\{Q, q_1\}$, and our cost is $p_1' q_1'$, where p_1' is the price corresponding to q_1'. □

3.1 Reverse Auctions with Non-Discriminatory Pricing

Consider a reverse auction where a buyer wants Q units of an item. Each of the n sellers submits a piecewise linear supply curve. In a *non-discriminatory* reverse auction, the buyer determines an optimum price p^* to minimize his cost, and sellers supply their share of units at price p^*. (The number of items purchased from seller i is computed using his supply curve, evaluated at price p^*.) The proof of the following theorem is similar to the proof of Theorem 2.4.

Theorem 3.3 *Consider a single-item, multi-unit reverse auctions with n bidders, each making a k-piece supply curve bid. Under non-discriminatory pricing, the auction can be solved so as to minimize cost in time $O(nk \log(nk))$ with or without free disposal.*

3.2 Reverse Auctions with Discriminatory Pricing

In a *discriminatory* reverse auction, the buyer determines for each seller j a distinct price p_j with the objective of minimizing the total cost $\sum_j p_j q_j$ subject to the supply constraint $\sum_j q_j \geq Q$. (Without free disposal, the quantity constraint is an equality.) The problem of clearing such an auction with piecewise linear supply curves again turns out to be $\mathcal{NP}$-Complete.

Theorem 3.4 *A single-item, multi-unit reverse auction with discriminatory pricing and piecewise linear (specifically step function) supply curve bids is $\mathcal{NP}$-Complete to clear so as to minimize cost (with or without free disposal).*

PROOF. The proof *without free disposal* is the same as in Theorem 2.5. The details of the *free disposal* case are different, so we discuss that here. We reduce the *knapsack* problem to the reverse auction, but since knapsack is a value-maximization problem, while the reverse auction is a cost-minimization problem, we need a minor transformation in between.

Let $\{(s_1, v_1), (s_2, v_2), \ldots, (s_n, v_n),\ K\}$ be an instance of the knapsack problem—the knapsack has capacity K, and item i has size s_i and value v_i. Let us create an instance of the single-item multi-unit reverse auction, as follows.

Bidder i places a step function bid (v_i, s_i), meaning he is willing to supply s_i units at lot price v_i (or at any price higher than v_i), and no units for a price less than v_i. Let $T = \sum_i s_i$ be the total number of units in all the bids. We set up the auction so that the buyer wishes to purchase at least $T - K$ units, at minimum possible price. Let S' be the set of bids that are winning bids in the reverse auction. Then, we claim that the remaining bids (those *not* in S') form a solution to the knapsack problem. First, since the bids in S' have a total of at least $T - K$ units, the remaining units are at most K, and so the solution is knapsack feasible. Second, since the bids in S' provide $T - K$ units at least possible price, the total price of the remaining bids is largest possible subject to the knapsack constraint. Thus, the auction problem is $\mathcal{NP}$-Complete. □

In fact, the reverse auction without free disposal is $\mathcal{NP}$-Complete *even with linear downward sloping supply curves*, unlike the seller auction with linear demand curves which is polytime solvable (except in the absurd case of constant demand lines and no free disposal), cf. Theorem 2.6.

Theorem 3.5 *A single-item, multi-unit reverse auction with discriminatory pricing and no free disposal is $\mathcal{NP}$-Complete to clear so as to minimize cost with downward sloping linear supply curve bids.*

PROOF. Consider an instance of the subset sum problem: set of non-negative integers $X = \{x_1, x_2, \ldots, x_n\}$, and an integer K. Corresponding to x_i, we create a linear supply curve $q = -p + x_i$. The subset sum problem has a solution if and only if the reverse auction without free disposal has a solution for quantity Q and cost 0. □

Remark. In the preceding theorem, the price zero is only for convenience. We can easily truncate the supply curves, for example at $p = 1$, to enforce a minimum unit price of one. In that case, the subset sum has solution if and only if the buyer can acquire exactly K units at total price K.

Remark. With free disposal and downward sloping or constant supply lines (that start at $p = 0$), the optimal solution is obtained in $O(n)$ time by accepting all supply lines at $p = 0$. If the aggregate quantity is at least Q, then the solution is feasible and optimal. Otherwise, no solution is feasible.

Remark. With constant supply lines and no free disposal, finding a feasible solution to the reverse auction is $\mathcal{NP}$-Complete because that corresponds to the *subset sum* problem. If a feasible solution exists, and the lines start at $p = 0$, then the optimal solution has zero cost (accept at $p = 0$ each one of the lines that are part of the feasible solution).

Remark. In reverse auctions, downward sloping supply corresponds to quantity discounts. A more common case would be *upward sloping supply*, i.e., the higher the price, the more the supplier is willing to sell. We analyze upward sloping supply lines in another paper [Sandholm and Suri, 2001], showing clearing complexity of $O(n \log n)$—the same complexity as *auctions* with *downward* sloping demand lines.

4 Price-Quantity Pair Bids

In this section we consider auctions where the auctioneer has multiple indistinguishable units of one item to sell, and bidders express their preferences via *price-quantity pairs*.

Definition 1 *In a* price-quantity pair *bid (p, q), the bidder states a price $p \in Z^+$ that he is willing to pay for $q \in Z^+$ units.*[2] *The bid is atomic, meaning it must be accepted as a whole or rejected—it cannot be accepted fractionally.*

Theorem 4.1 (Known) *If the seller has Q units, and each buyer submits one price-quantity bid, then the problem of maximizing revenue is equivalent to the $\mathcal{NP}$-Complete knapsack problem. It is solvable in pseudo-polynomial time $O(n^2V)$ [Garey and Johnson, 1979], where n is the number of bids and V is the maximum price of any bid. It is approximable within a $(1 - \varepsilon)$ factor of the optimum in polynomial time $O(n \log \frac{1}{\varepsilon} + \frac{1}{\varepsilon^4})$ [Lawler, 1976].*

[2] In this section we consider the case where p is the price for the entire lot q. If a unit price is stated instead, it can be trivially converted to a lot price by multiplying by q.

If there is *free disposal*, the auctioneer can always sell fewer than Q units because, at worst, he can dispose of the extra units for free. On the other hand, for many real goods there is no free disposal.

Theorem 4.2 *If the seller has to sell exactly Q units (or none if the instance is infeasible), and each buyer submits one price-quantity bid, then even finding a feasible solution is $\mathcal{NP}$-Complete. The problem of maximizing revenue can be solved optimally in pseudo-polynomial time.*

PROOF. Finding a feasible solution is $\mathcal{NP}$-Complete because the special case where $p_i = q_i$ for all i is equivalent to the *subset sum* problem, which is $\mathcal{NP}$-Complete. The problem can be solved in pseudo-polynomial time using a straightforward dynamic program. We omit it due to limited space. □

In many settings, a bidder could accept alternative quantities at different prices. This can be enabled by allowing each bidder to submit multiple price-quantity bids which are combined with XOR.

Theorem 4.3 *If a seller can sell at most Q units, and each buyer submits a set of alternative price-quantity bids (i.e., XOR bids), then the problem is $\mathcal{NP}$-Complete. It can be solved in pseudo-polynomial time $O(n^2VL)$, where n is the number of bidders, L is the maximum number of alternative bids by any buyer, and V is the maximum price of any bid. The problem can also be approximated within a $(1 - \varepsilon)$ factor of the optimum in $O(nL \log \frac{1}{\varepsilon} + \frac{1}{\varepsilon^4})$ time.*

PROOF. $\mathcal{NP}$-Completeness follows from Theorem 4.1 since each bidder might only submit one price-quantity pair.

We now devise a pseudo-polynomial algorithm. We label the bidders (arbitrarily), 1 through n. Observe that nV is an upper bound on the revenue. Let $A(i, v)$ denote the smallest number of units that can be sold to bidders in the set $\{1, 2, \ldots, i\}$ with total revenue exactly v.

1. [**Initialize:**] Let the alternative price-quantity bids of buyer 1 be (q_1, p_1) XOR (q_2, p_2) XOR $\ldots$ (q_j, p_j). Set
   - $A(1, p_t) = q_t$, for $t = 1, 2, \ldots, j$.
   - $A(1, v) = \infty$, for all other values of v.
2. **for** $i = 2$ to n
   for $v = 1$ to nV
   $A(i, v) = \infty$
   if buyer i has bid (q_1, p_1) XOR (q_2, p_2) XOR $\ldots$ (q_j, p_j), then
   for $t = 1$ to j
   if $p_t \leq v$ **then**
   $$A(i, v) = \min \left\{ \begin{array}{l} A(i, v), \\ A(i-1, v), \\ q_t + A(i-1, v - p_t) \end{array} \right\}$$
   else $A(i, v) = A(i - 1, v)$.

This algorithm computes a solution in $O(n^2VL)$ time. This pseudo-polynomial algorithm can be converted to an ε-approximation algorithm with running time $O(nL \log \frac{1}{\varepsilon} + \frac{1}{\varepsilon^4})$ using the scheme of Lawler [Lawler, 1976]. □

While XOR bids are fully expressive in the sense that they allow a bidder to express any valuation (mapping from the number of units to a price), in many cases, a more compact (and never less compact) representation of the same valuation can be obtained using the OR-of-XORs bidding language[3]. An

[3] XOR bids and OR-of-XORs bids were originally introduced for combinatorial auctions where there are multiple distinguish-

Market type	Linear curves			Piecewise linear curves
	Upward sloping	constant	downward sloping	
Nondiscriminatory auction	$O(n)$	$O(n)$	$O(n \log n)$	$O(nk \log(nk))$ *
Discriminatory auction	$O(n)$	fd: $O(n)$ nfd: $\mathcal{NP}$-Complete	$O(n \log n)$ *	$\mathcal{NP}$-Complete *
Nondiscriminatory reverse auction	$O(n \log n)$	$O(n)$	$O(n)$	$O(nk \log(nk))$ *
Discriminatory reverse auction	$O(n \log n)$ * [Sandholm and Suri, 2001]	fd: $O(n)$ nfd: $\mathcal{NP}$-Complete	fd: $O(n)$ nfd: $\mathcal{NP}$-Complete	$\mathcal{NP}$-Complete *

Table 1: Summary of our results on clearing supply/demand curves (fd = free disposal, nfd = no free disposal). The nontrivial results are marked with a "*".

OR-of-XORs bid is a bid where multiple XOR price-quantity bids are offered and any number of these can be accepted (subject to honoring the overall quantity Q):

$$[(q_1, p_1) \text{ XOR } (q_2, p_2) \text{ XOR } \ldots (q_i, p_i)] \text{ OR}$$
$$[(q_{i+1}, p_{i+1}) \text{ XOR } (q_{i+2}, p_{i+2}) \text{ XOR } \ldots (q_j, p_j)] \text{ OR} \ldots$$
$$[(q_k, p_k) \text{ XOR } (q_{k+1}, p_{k+1}) \text{ XOR } \ldots (q_l, p_l)].$$

Theorem 4.4 *If the auctioneer can sell at most Q units, and each bidder submits an* OR*-of-*XORs *bid, then the problem can be solved and approximated with the time complexities stated in Theorem 4.3, where n now is the number of* XOR*-disjuncts submitted overall, and L is the maximum number of bids within any* XOR*-disjunct.*

PROOF. We can treat different XOR bids from the same bidder as coming from different bidders. Since each bidder can be awarded any number of OR bids, this transformation is sound. We can thus assume that each bidder has submitted only one XOR bid, and solve the problem using the algorithm described in the proof of Theorem 4.3. □

Now, consider XOR bids and OR-of-XORs bids in settings where the auctioneer has to sell exactly Q units. It follows from Theorem 4.2 that finding a feasible solution (and therefore also approximation) is $\mathcal{NP}$-Complete (unlike in the free disposal setting). The problem can be solved in pseudo-polynomial time using a dynamic program akin to the ones above where the auctioneer can keep any of the units (we omit these due to limited space).

5 Conclusions

Market mechanisms play a central role in AI as a coordination tool in multiagent systems, and as an application area for algorithm design. Market mechanisms where buyers are directly cleared with sellers, and thus do not require an external liquidity provider, are highly desirable for electronic marketplaces for several reasons. In this paper we studied the inherent complexity of, and designed algorithms for, clearing auctions and reverse auctions with multiple indistinguishable units for sale.

Table 1 summarizes our results on market clearability under demand/supply curves. Note that in non-discriminatory settings, even when each bidder's curve is linear, the aggregate curve might not be linear but piecewise linear (because the bidder's curves have to be ignored below zero quantity).

In addition to the results summarized in the table, we believe that one of the most surprising result of our paper is the following property of discriminatory auctions: at the unconstrained optimum, each bidder generally gets a different price, but interestingly, to accommodate the constraint of limited supply, each bidder's price is incremented *equally* from the unconstrained optimum.

When bidders express their preferences with price-quantity pairs, the market clearing problem is essentially equivalent to the knapsack problem, and therefore $\mathcal{NP}$-Complete but solvable in pseudo-polynomial time. With free disposal, the problem admits a polynomial-time approximation scheme, but no such approximation scheme is possible without free disposal. We also describe pseudo-polynomial algorithms for XOR bids and OR-of-XORs bids, and their polynomial approximability when free disposal is allowed.

Our $\mathcal{NP}$-completeness results carry over to **exchanges** (where the objective is to maximize surplus, i.e., sum of accepted bids minus sum of accepted asks) directly since auctions and reverse auctions are special cases of exchanges.

Our algorithms help pave the way toward automated electronic markets without external liquidity providers, but at the same time, our $\mathcal{NP}$-Completeness results curtail the space of automated market designs that are computationally tractable.

able items for sale, and bids can be submitted on combinations of items [Sandholm, 1999; 2000]

References

[Fujishima *et al.*, 1999] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJCAI*, pages 548–553.

[Garey and Johnson, 1979] Michael R Garey and David S Johnson. *Computers and Intractability*. W. H. Freeman and Company.

[Lawler, 1976] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston.

[Lupien and Rickard, 1997] William A Lupien and John T Rickard. Crossing network utilizing optimal mutual satisfaction density profile. US Patent 5,689,652, granted Nov. 18 to Optimark Technologies.

[Sandholm and Suri, 2000] Tuomas Sandholm and Subhash Suri. Improved algorithms for optimal winner determination in combinatorial auctions and generalizations. In *AAAI*, pages 90–97.

[Sandholm and Suri, 2001] Tuomas Sandholm and Subhash Suri. Computational complexity of clearing exchanges. Technical report, 2001. In progress.

[Sandholm, 1999] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *IJCAI*, pages 542–547. First appeared: Washington Univ., Dept. of Computer Science WUCS-99-01, Jan. 28th.

[Sandholm, 2000] Tuomas Sandholm. eMediator: A next generation electronic commerce server. In *AGENTS*, pages 73–96, Barcelona, Spain, June 2000. Early version: AAAI-99 Workshop on AI in Electronic Commerce, pp. 46–55, July 1999, and Washington Univ., St. Louis, Dept. of Computer Science WU-CS-99-02, Jan. 1999.

[Tennenholtz, 2000] Moshe Tennenholtz. Some tractable combinatorial auctions. *AAAI*.

[Wurman *et al.*, 1998] Peter R Wurman, Michael P Wellman, and William E Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *AGENTS*, pages 301–308.

On Market-Inspired Approaches to Propositional Satisfiability

William E. Walsh
University of Michigan AI Laboratory
1101 Beal Ave, Ann Arbor, MI
48109-2110, USA
wew@umich.edu

Makoto Yokoo
NTT Communication Science Laboratories
2-4 Hikaridai, Seika-cho, Soraku-gun,
Kyoto 619-0237, Japan
yokoo@cslab.kecl.ntt.co.jp

Katsutoshi Hirayama
Kobe University of Mercantile Marine
5-1-1 Fukaeminami-machi, Higashinada-ku,
Kobe 658-0022, Japan
hirayama@ti.kshosen.ac.jp

Michael P. Wellman
University of Michigan AI Laboratory
1101 Beal Ave, Ann Arbor, MI
48109-2110, USA
wellman@umich.edu

Abstract

We describe two market-inspired approaches to propositional satisfiability. Whereas a previous market-inspired approach exhibited extremely slow performance, we find that variations on the pricing method with a simplified market structure can improve performance significantly. We compare the performance of the new protocols with the previous market protocol and with the distributed breakout algorithm on benchmark 3-SAT problems. We identify a tradeoff between performance and economic realism in the new market protocols, and a tradeoff between performance and the degree of decentralization between the new market protocols and distributed breakout. We also conduct informal and experimental analyses to gain insight into the operation of price-guided search.

1 Introduction

Agents must often engage in activities with complex, interrelated dependencies. Even finding a satisficing solution for problems such as resource allocation, scheduling, and production in a supply chain is often intractable for a central problem solver with global knowledge. The problem is further complicated when decentralization constraints such as locality of interest, knowledge, communication, and authority must be respected.

Yokoo and Hirayama [2000], [Yokoo, 2000] formalize such problems as distributed constraint satisfaction problems (DisCSPs) and, with others, have designed a variety of effective algorithms. These approaches are generally distributed adaptations of centralized algorithms. Recent interest in market-based approaches to distributed decision making, and open questions about the computational power of markets [Shoham and Tennenholtz, to appear], prompted Walsh and Wellman [2000] to apply a market-based supply chain formation protocol [Walsh and Wellman, 1998] to a 3-SAT reduction of the supply chain formation problem, an approach they called MarketSAT. They found that, although market prices can guide decentralized search, the approach was impractically slow.

In this paper we present alternate, simpler market-inspired approaches that provide more satisfactory performance, while respecting well-defined decentralization constraints. We evaluate the protocols on benchmark propositional satisfiability (SAT) problems. As the fundamental NP-complete problem, formally equivalent to a large class of combinatorial problems, SAT serves as a convenient problem class on which to systematically evaluate our protocols.

In Section 2 we introduce two new MarketSAT protocols with qualitatively different pricing mechanisms. In Section 3 we provide an economic interpretation of the protocols and discuss the rationality of the assumed agent behavior. In Section 4 we convey our understanding of price-guided search and in Section 5 we show that the protocols are incomplete. In Section 6 we compare the performance of the new pro-

tocols with the original MarketSAT and with the distributed breakout (DB) algorithm [Yokoo and Hirayama, 1996]. We also perform further experiments to explain the operation of the protocols. In Section 7 we compare the decentralization of the MarketSAT protocols relative to distributed breakout. We conclude in Section 8.

2 MarketSAT Protocols

Following the notation of Walsh and Wellman [2000], we consider propositional satisfiability problems with variables U and clauses Q, each containing sets of literals over U in conjunctive normal form (CNF). A clause is satisfied if at least one literal in the clause evaluates to T. The problem is to determine whether there exists a truth assignment $t : U \rightarrow \{T,F\}$ that satisfies each $q \in Q$.

A variable u ***fails to satisfy*** a clause q under truth assignment t iff either: (1) $t(u) = T$, $u \notin q$, and $\bar{u} \in q$, *or*, (2) $t(u) = F$, $\bar{u} \notin q$, and $u \in q$. A MarketSAT economy consists of agents representing variable assignments and a set of goods specifying licenses to fail to satisfy clauses. Agents choose assignments for their corresponding variables, but must acquire the necessary licenses for their chosen truth assignments. Clearly, $q \in Q$ is satisfied under t iff *at most* $|q|-1$ variables in q fail to satisfy q. Hence, we make available $|q|-1$ such licenses.[1] It follows that a problem is satisfiable iff all agents can choose assignments such that they can obtain all necessary licenses to support their assignments.

Observe that our notion of failing to satisfy a clause is equivalent to the notion of a nogood in CSPs, which in SAT is just a negated clause. MarketSAT could be applied to more general CSPs by specifying the licenses to correspond to nogoods, with $|q|-1$ licenses available for each nogood of size $|q|$.

In a MarketSAT protocol, the agents search for a satisfying solution in a decentralized fashion by negotiating to acquire the licenses. A market protocol consists of an auction mechanism and agent bidding policies. The negotiation for each license type is mediated by a separate auction. Although the two new MarketSAT protocols differ in the auction rules and bidding policies, they share common high-level features. A MarketSAT protocol iterates through the following steps:

1. Agents select tentative truth assignments and submit ***bids*** to a subset of the auctions corresponding to their chosen truth assignments.
2. Auctions send ***price quote*** messages to the agents indicating the current going prices of the licenses.

The protocol continues until ***quiescence***, a state in which no agent chooses to update any of its bids or the auctions terminate negotiation according to certain protocol-specific conditions. The auctions allocate their respective licenses to agents when quiescence is reached. For our empirical studies we assume synchronous instantaneous communication. Nothing in our protocols requires synchrony, but this allows us to fix certain parameters, focusing our studies.

The present market structure is a simplification of the original MarketSAT economy [Walsh and Wellman, 2000] based on a supply chain formation model [Walsh and Wellman, 1998]. In the original economy, there was a separate agent for positive and negative assignments of variables. An auction for each variable mediated the negotiation of the assignment agents to provide an assignment for the respective variables. An end consumer bid to acquire an assignment to each variable. Experiments show that the new protocols perform much faster with the new structure (Section 6).

In the following sections we describe the details of two new variants of MarketSAT.

2.1 Uniform Pricing

We describe a MarketSAT protocol with uniform pricing (MS-U) based on the original MarketSAT protocol (MS-O), but adapted to the present simplified structure. An auction allows an agent to place a bid to buy a license at a specified price. An agent may update its bid only if it increases the price of the bid by at least some publicly known increment δ. Agents may not withdraw bids.

Given a set of buy bids and $|q|-1$ units of license g available (for the right to fail to satisfy a clause q), an auction reports a price quote comprised two parts: 1) the ***bid price***, $\beta(g)$, is the $(|q|)$th highest price of all bids, and 2) the ***ask price***, $\alpha(g)$, is the $(|q|-1)$st highest price of all bids. If there are fewer than $|q|-1$ bids in the auction, the bid and ask prices are zero. If there are $|q|-1$ bids in the auction, the bid price is zero and the ask price equals the price of the lowest bid. The bid price specifies the price that the winning bidders would pay if the auction stopped in the current state and the ask price specifies how much a losing agent must bid in order to be a winning bidder. The price quote also indicates to a bidder whether its bid is one of the $|q|-1$ highest (winning) bids. In quiescence, an auction allocates its units of license g according to the (M+1)st price auction rules [Wurman *et al.*, 1998]—the agents with the $|q|-1$ highest bids win g and pay $\beta(g)$. For the price quotes and final allocation, the auction breaks ties in favor of earlier received bids, and randomly between simultaneously received bids.

An agent randomly chooses the initial assignment for its variable and places bids at price zero for the necessary licenses. When it subsequently receives price quotes, it chooses an assignment that minimizes its assignment cost, defined for an assignment as the maximum of the sum of its current ***perceived costs*** for the licenses needed for that assignment, and the previously computed assignment cost. In the case of ties, the agent keeps its current assignment. An agent's perceived cost for license g is $\beta(g)$ if it is winning, $\alpha(g)$ if it has not submitted a bid, $\alpha(g)$ if it is both losing and $\alpha(g) > \beta(g)$, and $\alpha(g)+\delta$ if it is both losing and $\alpha(g) = \beta(g)$.

After an agent chooses its assignment, it increments by δ any losing bid it has sent to the auction, if it needs the license for its chosen assignment. If it hasn't submitted a bid for some needed license, it submits a bid at price zero. An agent does not update its bids if it is winning all the licenses necessary for its current assignment. Observe that if the market

[1] In the original MarketSAT implementation [Walsh and Wellman, 2000], we used a reduction from 3-SAT and made 2 units of each license available. The existence of $|q|-1$ licenses for a clause of size $|q|$ is a straightforward generalization.

reaches quiescence in this way, then the agents' local assignments constitute a globally satisfying assignment.

2.2 Differential Pricing

In this section we describe a MarketSAT protocol with differential pricing (MS-D). An auction allows an agent to place a bid to demand either either zero or one units of the license, without specifying a price. Agents may switch between zero and one quantity demands without constraint.

An auction may report different price quotes to each bidder, depending on the demand for the license. An auction maintains a nondecreasing ***premium price*** for its license. If the auction has $|q|-1$ units of the license available, and the total demand expressed by the bids is d, then the auction reports price quotes as follows:

- If $d < |q|-1$, then the auction reports a price of zero to all bidders.
- If $d = |q|-1$, then the auction reports a price of zero to all bidders with demand of one, and reports the premium price to the single bidder with demand of zero.
- If $d > |q|-1$, then the auction increases the premium by one, reports the premium price to one randomly chosen bidder with a demand of one, and reports a price of zero to all other bidders with demand of one.

In quiescence, if $d > |q|-1$, then the auction randomly allocates the license to $|q|-1$ agents that bid with demand one for the license, otherwise it allocates a license to each of the current bidders with demand one. Each agent that receives a license pays according the price specified in the last price quote it received.

An agent randomly chooses the initial assignment for its variable. When it subsequently receives price quotes, it chooses an assignment that minimizes the sum of the prices of licenses as reported by the last price quote, with preference for its current assignment when the costs are equal. When an agent is in its initial state, or when it flips its assignment, it places bids of quantity one for all licenses it needs for the assignment and places bids for quantity zero for all licenses necessary for the opposite assignment. When an agent does not flip its assignment, it resubmits any bid needed for that assignment and for which it received a premium price quote. If an agent does not flip its assignment and if received a zero price quote for all its bids for the assignment, the agent does not update any of its bids. Observe that if the market reaches quiescence in this way, then the agents' local assignments constitute a globally satisfying assignment. Furthermore, in this case every agent pays zero for the licenses it receives.

3 Economic Interpretation and Rational Behavior

The interpretation of the market-inspired protocols in economic terms requires a model of agent values under which the assumed behavior would be plausibly rational. For an agent a to be willing to participate in these protocols, and hence be willing to pay money for their allocations, it must obtain some individual value v_a from participating in a satisfying solution. An agent obtains no value if it does not participate in a satisfying solution. An agent wishes to maximize its surplus value, which is the difference between the value it obtains and the total price it pays for the licenses it acquires.

In both protocols, the bidding policies are non-strategic in that agents do not account for the behavior of other agents. This assumption may be reasonable in large networks for which agents have little information about other agent preferences and behavior.

The bidding policies are myopic and best-response in that agents always bid to optimize their surplus given the current price quotes, without speculating about future price changes. The plausibility of this approach depends on how accurately the price quotes indicate the prices agents may actually have to pay for their final allocations. For both protocols, the price quotes indicate what the agents would pay if the bidding stopped in the current state.

However, the protocols differ significantly in how price quotes signal future price movements. In MS-U, the nondecreasing price quotes indicate lower bounds on the amount that agents would have to pay, providing a basis for agents' belief in the price quotes. Since prices do not decrease, we would also expect that, in addition to the agent quiescence condition specified in Section 2.1, a rational agent would stop bidding when the total cost of an assignment exceeds v_a.

In MS-D, agents actually pay nothing for a globally satisfying allocation, and would have positive payments only if the protocol were to terminate with an unsatisfying allocation. Thus, unlike in MS-U, we would expect rational agents to stop bidding only when they reach a satisfying allocation (as specified in Section 2.2). But this calls into question the usefulness of the price quotes as meaningful future price indicators. The bidding policies in MS-D would be more plausibly rational if the agents had a reasonable expectation that the protocol could terminate at any time, for instance if the auctions terminated negotiations based on a random signal. Indeed, such a mechanism may be useful both to encourage desirable behavior and to bound the length of negotiations.

```
Initialize the weight of all nogoods to 1
UNTIL current state is solution DO
      IF current state is not a local minimum
      THEN make any local change that reduces
            the total weights of violated nogoods
      ELSE increase weights of all currently
            violated nogoods
END
```

Figure 1: The breakout algorithm [Morris, 1993].

4 Price-Guided Search

Intuitively, market prices indicate the relative global value of licenses, and agents use the prices to guide their local decisions. The bidding process can be seen as a distributed search for prices that support a satisfying allocation. In the MarketSAT protocols, prices are closely analogous to the weights of nogoods (which correspond exactly to failing to satisfy a clause) in the breakout algorithm [Morris, 1993], presented in Figure 1. In breakout, weights are a measure of how often

the nogoods appear in local minima visited by the algorithm. The weights thus bias the search away from local minima.

In contrast to breakout, which increases the costs of nogoods only when the global state is a local minimum, the MarketSAT protocols increase the prices of licenses in response to local state. The price of a license increases whenever the corresponding clause is not satisfied for the current assignment, which can and does occur outside of local minima of the global state.

The breakout algorithm sequentially flips variable assignments to reduce the global weight of the violated nogoods. The nogood weights are counted only for clauses that are not satisfied. In MS-U, because bids cannot be withdrawn, once a license has an excess demand, it will always have excess demand, even if the current choices of variable assignments would indicate that the clause is currently satisfied. Thus the prices never decrease and agents attribute a cost to a license even if the clause is satisfied. In contrast, in MS-D an agent will only attribute a positive cost to a license if either the license is overdemanded or if flipping its own assignment would make the license overdemanded (assuming no other agent would also flip). In this sense, the cost evaluation in MS-D is closer to the breakout algorithm than in MS-U.

In breakout, current nogood costs are evaluated uniformly for all variables. In MS-U, an agent distinguishes its cost evaluation depending on whether it is winning or losing and assumes that its total cost over all licenses never decreases. This difference in perceived prices gives extra "friction" to the currently winning agents because they assume lower costs, hence are somewhat less likely to swap assignments. However, this friction is relatively small because agents increase their bids by δ only when they are losing, hence the bid and ask prices never differ by more than δ. In MS-D, agents can attribute widely varying costs to a license, with at most one agent attributing the premium cost and others attributing a zero cost.

Unlike breakout, variables can flip simultaneously in the MarketSAT protocols. This could be beneficial to performance when agents operate in parallel, if the right flips occur simultaneously. Yokoo and Hirayama [1996] observed that it could be detrimental if ***neighbor*** variables—those that share a clause—flip simultaneously, and thus incorporated synchronizing steps into DB to prevent simultaneous neighbor flips.

Of particular concern are ***simultaneous, satisfying neighbor flips***—neighbors simultaneously flipping to satisfy a clause (e.g., variables x and y simultaneously flipping to T to satisfy clause $(x \vee y)$) which we expect to be prevalent in the MarketSAT protocols. When neighbor variables simultaneously flip to satisfy the same clause, other clauses may become needlessly unsatisfied. Unlike DB, the MarketSAT protocols have no explicit mechanism to prevent simultaneous neighbor flips, yet we expect the different protocols to differ in the number of simultaneous, satisfying neighbor flips. In MS-U, because the bid/ask spread on any license is small, agents that desire common licenses are likely to have similar total cost evaluations. Hence, when the price of a shared license increases, they may be likely to simultaneously flip to satisfy the clause. In MS-D, because agents can attribute widely varying costs to licenses, the cost evaluations of neighbor agents are less likely to be close than in MS-U. We expect this would reduce the number of simultaneous, satisfying neighbor flips.

5 Completeness

We can show that MS-U is incomplete using a close variant of the example that Morris [1993] used to show that the (centralized) breakout algorithm is incomplete. We assume synchrony, noting that this implies the same for an asynchronous system. The CNF clauses are: $(\bar{x} \vee y)$, $(\bar{x} \vee z)$, $(\bar{x} \vee w)$, $(\bar{y} \vee x)$, $(\bar{y} \vee z)$, $(\bar{y} \vee w)$, $(\bar{z} \vee x)$, $(\bar{z} \vee y)$, $(\bar{z} \vee w)$, $(\bar{w} \vee x)$, $(\bar{w} \vee y)$, $(\bar{w} \vee z)$. Observe that the only solution for the problem assigns all variables T or all variables F. Consider the case when the initial random assignments are $t(x) = t(z) = T$ and $t(y) = t(w) = F$. Assume that the random tie breaking occurs as follows in the first round: x wins $(\bar{x} \vee y)$, y wins $(\bar{z} \vee y)$, z wins $(\bar{z} \vee w)$, and w wins $(\bar{x} \vee w)$. Assume that the random tie breaking occurs as follows in the second round: y wins $(\bar{y} \vee x)$, x wins $(\bar{w} \vee x)$, z wins $(\bar{y} \vee z)$, and w wins $(z \vee \bar{w})$. In all subsequent rounds, auction tie breaking is deterministic. Throughout, all agents flip simultaneously, hence the protocol oscillates indefinitely.

With the same SAT instance we used to show incompleteness of MS-U (but with different initial assignments and tie breakings), our simulation results strongly suggest to us that MS-O is not guaranteed to converge to a solution. We refrain from describing a trace of a non-converging run due to the greater complexity of the agent interactions in MS-O.

We can also show that MS-D is incomplete using Moris's exact example. The CNF clauses are: $(x \vee y \vee z \vee w)$, $(\bar{x} \vee y)$, $(\bar{x} \vee z)$, $(\bar{x} \vee w)$, $(\bar{y} \vee x)$, $(\bar{y} \vee z)$, $(\bar{y} \vee w)$, $(\bar{z} \vee x)$, $(\bar{z} \vee y)$, $(\bar{z} \vee w)$, $(\bar{w} \vee x)$, $(\bar{w} \vee y)$, $(\bar{w} \vee z)$. Observe that the only solution assigns all variables T. Consider the initial random assignment t, such that $t(x) = T$ and all other variables are F. The premium increases in each of the unsatisfied clauses so long as they remain unsatisfied, but the choice of which variable in the clauses pays the premium is chosen randomly. With positive probability, x can always be the only variable chosen to pay the premium, in which case it will flip indefinitely and no other variables will flip.

If the breakout algorithm reaches truth assignment t as described in the preceding, it *cannot* converge to the satisfying truth assignment [Morris, 1993]. But because MS-D has variable-specific pricing, it does not necessarily make a flip when it would be a global improvement, which can delay undesirable flips. In fact, MS-D can always converge, with positive probability, to some satisfying truth assignment t^*. With positive probability, an auction for a license corresponding to an unsatisfied clause will always report the premium price to some variable x such that $t(x) \neq t^*$. But then the protocol would clearly converge to t^*. We do not know if a similar result holds for MS-U.

6 Experiments

We compared the MarketSAT protocols with MS-O and DB using satisfiable (unforced, filtered) uniform random 3-SAT problems at the phase transition (between 4.26 and 4.3

clause/variable ratio) from the SATLIB benchmark library[2]. This is generally considered the hardest class of 3-SAT problems to solve [Cheeseman *et al.*, 1991; Mitchell *et al.*, 1992] and has been used widely to benchmark SAT algorithms. Note that, to generate hard satisfiable problems, it is important to generate problems randomly and filter the unsatisfiable instances. Forcing the problems to satisfy a particular assignment makes them much easier [Achlioptas *et al.*, 2000].

To bound the computational time, for a problem instance of n variables, we ran a protocol for at most $1000n$ rounds. For MarketSAT, a round is a synchronous bidding round, and for DB, a round corresponds to a cycle as described by Yokoo and Hirayama [1996]. Agents participate until they reach quiescence (with a satisfying solution), or until the maximum rounds is reached. We recognize that the artificial limit on the bidding rounds reduces the plausibility that agents may use the specified bidding policies (see Section 3), but chose this approach to consistently compare the performance of the protocols. We compared these protocols in terms of bidding rounds, which is the best measure of performance when agents operate in parallel. The performance is summarized in Table 1. Comparing the protocols from top to bottom, each subsequent protocol improves by about a factor of two to seven compared to the previous on all performance measures.

n	Success Ratio	Average Rounds	Median Rounds	σ Rounds
		Protocol: MS-O		
50	0.40	3.90×10^4	5.00×10^4	18.1×10^3
		Protocol: MS-U		
50	0.96	6.12×10^3	1.51×10^3	1.15×10^4
75	0.75	2.75×10^4	1.08×10^4	3.03×10^4
100	0.53	6.10×10^4	8.21×10^4	4.16×10^4
		Protocol: MS-D		
50	1.00	896	250	3.52×10^3
75	0.98	3.98×10^3	429	1.17×10^4
100	0.96	1.04×10^4	1.50×10^3	2.34×10^4
125	0.85	2.74×10^4	3.65×10^3	4.45×10^4
150	0.85	3.69×10^4	5.94×10^3	5.45×10^4
175	0.83	5.37×10^4	1.63×10^4	6.68×10^4
		Protocol: DB		
50	1.00	234	64.5	829
75	0.99	2.14×10^3	299	9.40×10^3
100	0.98	4.26×10^3	460	1.61×10^3
125	0.96	9.12×10^3	1.42×10^3	2.63×10^4
150	0.93	1.80×10^4	1.22×10^3	4.24×10^4
175	0.88	2.98×10^4	2.83×10^3	5.81×10^4

Table 1: Performance of protocols with n variables.

We attempted to determine the causes of differing performance of the protocols. We believe that the improvement of MS-U over MS-O is due to the simplified network structure, since MS-U is otherwise essentially the same as MS-O. One plausible conjecture for the difference between MS-U, MS-D, and DB is differing numbers of simultaneous, satisfying neighbor flips. We predict that an ordering of the protocols by increasing simultaneous, satisfying neighbor flips would be the same as by increasing performance.

To test this conjecture we measured the simultaneous, satisfying neighbor flips in both MS-D and MS-U. Also, recognizing that the MarketSAT protocols are much like breakout, but with simultaneous flips, we modified the breakout algorithm to allow simultaneous flips (with some ad hoc parameters to control the number of simultaneous flips and simultaneous, satisfying neighbor flips). To gain insight about the effects of simultaneous, satisfying neighbor flips, we tested breakout with simultaneous flips (SFB) with varying fractions of such neighbor flips. We also varied the number of generic simultaneous flips to help distinguish their contribution to performance from neighbor flips.

Protocol	Rounds	Flips	Flips Per Flip Round	Neighbor Flips Ratio
MS-U	6.12×10^3	1.02×10^4	2.21	0.33
MS-D	896	473	2.02	0.23
DB	234	218	1.41	0
SFB	483	329	1.00	0
	403	374	1.66	0
	494	390	1.42	0.02
	474	413	1.53	0.10
	483	432	1.65	0.13
	410	340	1.66	0.23
	408	468	1.99	0.37

Table 2: Comparison of simultaneous, satisfying flips for 50-variable problems.

Table 2 shows the average number of flips, flips per flip round (flips per round in which a flip actually occurred) and average fraction of simultaneous, satisfying neighbor flips for MS-U, MS-D, DB, and SFB. As expected, MS-U performs substantially more simultaneous, satisfying neighbor flips absolutely, and as a fraction of the total flips than does MS-D. However, the results from SFB do not support the conjecture that the neighbor flips contribute significantly to the performance difference. The performance of SFB does not appear to be sensitive to, or even monotonic in, the fraction of simultaneous, satisfying neighbor flips.

Table 2 suggests an alternate explanation of the performance difference between DB and MS-D. For 50-variable problems, MS-D requires nearly 3.8 times as many rounds as DB, but only 2.2 times as many flips. We also measured the number of rounds in which MS-D and DB performed flips and found found that, in fact, the average number of rounds in which variables actually flip in MS-D is 316, only 35% of the average total rounds. The average number of rounds in which DB performed a flip is 73% of its average total rounds. Thus it seems that the poor performance of MS-D relative to DB is due to the extra rounds to required to produce flips.

We conjecture that these extra rounds in MS-D happen be-

[2] `http://aida.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html`

cause auctions randomly reassign the premium price at each round. Thus the agents' costs fluctuate randomly, and do not directly progress every time the premium increases. To verify this conjecture, we tested a variation of MS-D whereby pricing is reported as the premium and a fraction b of the premium (rather than the premium or zero as in MS-D). With a positive b, an agent's costs would not fluctuate so heavily from random assignments of the premium cost. We tried $b = 0.1$ on 50-variable problems and found that it required only 509 rounds, 462 flips, and 231 flip rounds. The ratio of rounds to flip rounds is only 2.2 with $b = 0.1$, compared to 2.8 for MS-D. Furthermore, although the variant with $b = 0.1$ outperformed MS-D, the fraction of satisfying neighbor flips was 0.43—significantly more than in MS-D. This evidence suggests that difference in performance between MS-D and DB is largely due to the extra rounds required to produce flips in MS-D, rather than simultaneous, satisfying neighbor flips.

Appealing to extra, non-flipping rounds does not seem to explain the relative performance difference between MS-U and MS-D, as the ratio of rounds to flip rounds is only 1.3 for MS-U, significantly less than in MS-D. An alternate explanation we propose is that the relatively poor performance of MS-U is due to the fact that costs continue to be attributed to a license when its respective clause becomes satisfied (note that prices are *increased* only for unsatisfied clauses though). If the prices do not distinguish between satisfied and unsatisfied clauses, it would seem that MS-U pricing may not provide a effective indication of the relative difficulty of satisfying a clause. In contrast, recall that, in MS-D, when a clause is satisfied, the agents currently demanding the associated license receive price quotes of zero. Breakout attributes no cost to satisfied clauses. To test whether attributing costs to satisfied clauses can be detrimental, we modified the breakout algorithm so that it does just that and found that the algorithm rarely found satisfying assignments. Of course MS-U did not perform this poorly, but it does differ from breakout in other ways, as described in Section 4. Still, our test suggests that we have identified a significant cause of the relatively poor performance of MS-U.

7 Discussion of Decentralization

The MarketSAT protocols are highly decentralized in the sense that agents need only know about and communicate with auctions for their own licenses (which in turn requires knowledge about the clauses in which they are contained) and auctions need only communicate with the agents that participate in them. Agents need not communicate with, or even know the existence of other variables. Similarly, auctions need not communicate with each other. In DB, an agent must know in which clauses its variable is contained and must also communicate with all variables in those clauses. MarketSAT can operate fully asynchronously. In DB, variables synchronize with their neighbors to detect quasi-local minima and to ensure that neighbor variables do not flip simultaneously.

In their previous work, Walsh and Wellman [2000] suggested that the highly decentralized nature of MS-O necessarily engenders poor performance. However, our experiments with MS-D suggest that a highly decentralized market-inspired protocol can perform reasonably well. Indeed, our experiments with variants on MS-D suggest that further improvements can be obtained with the same degree of decentralization. Still, it is an open question whether decentralized approaches could perform as well as the centralized SAT algorithms. Centralized algorithms can utilize techniques such as restarts (which contributed significantly to the success of GSAT [Selman *et al.*, 1992] and subsequent hill-climbing based algorithms) to help reduce heavy tails in the performance distribution. We found that restarts can also significantly improve the performance of MS-D. For example, with $10n$ max flips and 1000 max restarts, MS-D can solve all 175-variable problem instances within the bound, with average rounds 2.69×10^4, median 1.1×10^4, and $\sigma = 5.00 \times 10^4$. Although restarts could be implemented in a synchronous system with cooperating agents it is not obvious how such techniques might be utilized in a distributed, asynchronous system. Moreover, we do not have any intuition for an economic interpretation of restarts.

8 Conclusions

We described two market-inspired protocols for propositional satisfiability and compared them with the distributed breakout algorithm. We found that the pricing method can significantly affect the performance, with the differential pricing protocol about a half magnitude better than uniform pricing. However, the differential pricing protocol is less justifiable in terms of rational economic agent behavior. Although the MarketSAT protocols we consider perform significantly better than the original MarketSAT, the less decentralized distributed breakout algorithm still outperforms the differential pricing MarketSAT by a factor of three to four.

An informal analysis suggests that the price-guided search of MarketSAT works because the protocols resemble the centralized breakout algorithm. We found that the fraction of simultaneous, satisfying neighbor flips does not explain the difference in performance across protocols, but that the performance of MarketSAT with differential pricing suffers largely from the extra rounds needed to produce a flip.

We have identified tradeoffs in terms of runtime performance, decentralization, and the plausibility of assumed agent behaviors. Understanding these tradeoffs is necessary to make informed engineering decisions about the appropriateness and applicability of alternate decentralized approaches to a particular problem environment.

The market approach has the benefit of providing a price-based interface for an agent to evaluate and direct its behavior in the context of its broader decision making. To better understand and further develop market approaches to complex coordination problems, we must explicitly incorporate a model of the agents' economic motivations in the context of the problem to be solved. Future work should also include a deeper analysis of rational agent behavior.

Acknowledgments

The basic ideas for this project were developed during a visit to the University of Michigan by the second author. The authors wish to thank NTT and Edmund H. Durfee for support-

ing the visit. The first author was supported by a NASA/JPL Graduate Student Researcher fellowship.

References

[Achlioptas *et al.*, 2000] Dimitris Achlioptas, Carla Gomes, Henry Kautz, and Bart Selman. Generating satisfiable problem instances. In *Seventeenth National Converence on Artificial Intelligence*, pages 256–261, 2000.

[Cheeseman *et al.*, 1991] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the *Really* hard problems are. In *Twelfth International Joint Conference on Artificial Intelligence*, pages 331–337, 1991.

[Mitchell *et al.*, 1992] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of SAT problems. In *Tenth National Conference on Artificial Intelligence*, pages 459–465, 1992.

[Morris, 1993] Paul Morris. The breakout method for escaping from local minima. In *Eleventh National Conference on Aritificial Intelligence*, pages 40–45, 1993.

[Selman *et al.*, 1992] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Tenth National Conference on Artificial Intelligence*, pages 440–446, 1992.

[Shoham and Tennenholtz, to appear] Yoav Shoham and Moshe Tennenholtz. On rational computability and communication complexity. *Games and Economic Behavior*, to appear.

[Walsh and Wellman, 1998] William E. Walsh and Michael P. Wellman. A market protocol for decentralized task allocation. In *Third International Conference on Multi-Agent Systems*, pages 325–332, 1998.

[Walsh and Wellman, 2000] William E. Walsh and Michael P. Wellman. MarketSAT: An extremely decentralized (but really slow) algorithm for propositional satisfiability. In *Seventeenth National Conference on Artificial Intelligence*, pages 303–309, 2000.

[Wurman *et al.*, 1998] Peter R. Wurman, William E. Walsh, and Michael P. Wellman. Flexible double auctions for electronic commerce: Theory and implementation. *Decision Support Systems*, 24:17–27, 1998.

[Yokoo and Hirayama, 1996] Makoto Yokoo and Katsutoshi Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Second International Conference on Multi-Agent Systems*, pages 401–408, 1996.

[Yokoo and Hirayama, 2000] Makoto Yokoo and Katsutoshi Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):198–212, 2000.

[Yokoo, 2000] Makoto Yokoo. *Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems*. Springer, 2000.

MULTI-AGENT SYSTEMS

MULTI-AGENT SYSTEMS

Achieving Budget-Balance with Vickrey-Based Payment Schemes in Exchanges

David C. Parkes
Computer and Information Science Department
University of Pennsylvania
200 South 33rd Street, Philadelphia, PA 19104
dparkes@unagi.cis.upenn.edu

Jayant Kalagnanam and **Marta Eso**
IBM T.J. Watson Research Center
P.O. Box 218,
Yorktown Heights, NY 10598
jayant@us.ibm.com, martaeso@us.ibm.com

Abstract

Generalized Vickrey mechanisms have received wide attention in the literature because they are efficient and strategy-proof, i.e. truthful bidding is optimal whatever the bids of other agents. However it is well-known that it is impossible for an exchange, with multiple buyers and sellers, to be efficient and budget-balanced, even putting strategy-proofness to one side. A market-maker in an efficient exchange must make more payments than it collects. We enforce budget-balance as a hard constraint, and explore payment rules to distribute surplus after an exchange clears to minimize distance to Vickrey payments. Different rules lead to different levels of truth-revelation and efficiency. Experimental and theoretical analysis suggest a simple *Threshold scheme*, which gives surplus to agents with payments further than a certain threshold value from their Vickrey payments. The scheme appears able to exploit agent uncertainty about bids from other agents to reduce manipulation and boost allocative efficiency in comparison with other simple rules.

Introduction

The participants in an exchange, or agents, can submit both *bids*, *i.e.* requests to buy items for no more than a bid price, and *asks*, *i.e.* requests to sell items for at least an ask price. Exchanges allow *multiple* buyers to trade with *multiple* sellers, with aggregation across bids and asks as necessary to clear the market. An exchange might also allow agents to express logical conditions across bundles of different items; for example, an agent might want to buy "A and B", or sell "A and B, or C". Following the literature on combinatorial auctions (Rothkopf *et al.* 1998; de Vries & Vohra 2000) we call this a *combinatorial exchange*. Applications of combinatorial exchanges have been suggested to excess steel inventory procurement (Kalagnanam *et al.* 2000) and to supply chain coordination (Walsh *et al.* 2000).

The market maker in an exchange collects bids and asks and clears the exchange by computing: (i) a set of trades, and (ii) the payments made and received by agents. In designing a mechanism to compute trades and payments we must consider the bidding strategies of self-interested agents, i.e. rational agents that follow expected-utility maximizing strategies. We take as our primary goal that of *allocative-efficiency*: to compute a set of trades that maximize value. In addition, we require:

–*individual-rationality* (IR), or voluntary participation, such that all agents have positive expected utility to participate.
– *budget-balance* (BB), such that the exchange does not run at a loss.

Another useful property is *incentive-compatibility* (IC), which states that truthful bidding (submitting bid and ask prices equal to an agent's value) forms a Bayesian-Nash equilibrium. In other words, every agent can maximize its expected utility by bidding its true values, given that every other agent also bids truthfully. A stronger condition is *strategy-proofness*, such that truthful bidding is optimal whatever the bids of other agents. Strategy-proofness is useful computationally because agents can avoid game-theoretic reasoning about other agents.

Unfortunately, the well-known result of Myerson & Satterthwaite (1983) demonstrates that *no* exchange can be efficient, budget-balanced (even in the average-case), and individual-rational. This impossibility result holds with or without incentive-compatibility[1], and even in Bayesian-Nash equilibrium. Instead, one must:
(a) impose BB and IR, and design a fairly efficient but incentive-compatible (or perhaps strategy-proof) scheme.
(b) impose BB and IR, and design a fairly efficient and fairly incentive-compatible scheme.

We follow (b), and design a mechanism for combinatorial exchanges (with multi-unit and regular exchanges as special cases) that promotes reasonable truth-revelation and reasonable allocative-efficiency. The mechanism computes the value-maximizing allocation given agent bids, and computes payments to reduce the utility for non-truthful bidding.

Earlier authors (Myerson & Satterthwaite 1983; McAfee 1992; Barbera & Jackson 1995) have followed approach (a), deliberately computing allocations that are inefficient for truthful bids from agents to achieve *incentive-compatibility* or strategy-proofness. We do not believe their schemes extend easily to combinatorial problems. Furthermore, we believe that our scheme is particularly useful with bounded-rational agents with incomplete information about other agents, because such agents are unable to fully exploit the "holes" for manipulation that remain in our designs.

A Vickrey-Based Payment Scheme

Our particular approach takes the Vickrey payment scheme, and adapts it to make it budget-balanced. Without the prob-

[1] As it must, by the revelation principle.

lem of BB, Vickrey payments support an efficient, IR, and strategy-proof exchange.

We interpret Vickrey payments as an assignment of discounts to agents after the exchange clears. BB is achieved so long as the market maker distributes no more than the available surplus when the exchange clears. The pricing problem is formulated as an optimization problem, to compute discounts to minimize the distance to Vickrey discounts. We derive the payment schemes that correspond to optimal solutions to a number of different distance functions.

Theoretical and experimental analysis compares the utility to an agent for misstating its value in bids and asks in each payment scheme across a suite of problem instances. The results, both theoretical and experimental, make quite a compelling argument for a simple *threshold* payment scheme which provides discounts to agents with payments more than a threshold distance than their Vickrey payments.

The Threshold rule increases the amount by which an agent with a large degree of manipulation freedom must adjust its bid to have a useful effect on the price it finally pays, while leaving unaffected the manipulation properties for agents with a small degree of manipulation freedom. The effect is to reduce manipulation and boost allocative-efficiency in comparison with other schemes.

Let us introduce an example problem, that we will return to later in the paper.

Example. Suppose agents 1, 2, 3, 4. Agents 1 and 2 want to sell A and B respectively, with values $v_1(A) = \$10$ and $v_2(B) = \$5$. Agents 3 and 4 want to buy the bundle AB, with values $v_3(AB) = \$51$ and $v_4(AB) = \$40$. The efficient allocation is for agents 1 and 2 to trade with agent 3, for a net increase in value of \$36.

The mechanism design problem is: given bid and ask prices for A, B and AB from the agents, what trades should take place and what payments should be made and received?

Vickrey Based Surplus Distribution

The market maker in an exchange has two problems to solve: *winner determination*, to determine the trades executed, and *pricing*, to determine agent payments. A common goal in winner-determination is to compute trades that maximize *surplus*, the difference between bid prices and ask prices.[2] These trades implement the efficient allocation with truthful bids and asks.

The pricing problem is to determine agent payments when the exchange clears. In this section we describe an application of the Vickrey-Clarke-Groves pricing mechanism (Vickrey 1961; Clarke 1971; Groves 1973) to an exchange, which often fails BB. The presentation is for a combinatorial exchange, in which agents can bid and ask for bundles of items and express logical constraints, e.g. "exclusive-or" and "additive-or" constraints, across bids and asks.[3]

[2]The payment schemes presented in this paper are also applicable with any (*ex ante* fixed) constraint on feasible trades; e.g. any level of aggregation in matching trades, or side constraints, e.g. on the volume of trade or degree of dominance by a single agent.

[3]Vickrey payments in exchanges for homogeneous items, with and without multi-unit bids can be derived as special cases (Wurman *et al.* 1998).

Computing payments in a Vickrey-based exchange also requires solving a number of winner-determination problems, once without each agent that trades. Winner-determination is NP-hard for general combinatorial exchange problems and intractable as problems become large. However, our current focus is on the incentive properties of novel Vickrey-based payment schemes. Tractable winner-determination is not our present concern. This noted, the payment schemes proposed are immediately applicable to tractable special cases of combinatorial exchanges (see Kalagnanam et al.) Future work should explore the effect of layering our schemes on top of approximate winner-determination algorithms.

We first define the Vickrey payments in an exchange, and then argue that the failure of BB is quite pervasive with Vickrey payments in exchanges.

Vickrey Payments

Let $\mathcal{L}$ denote the set of agents and $\mathcal{G} = \{A, B, C, \dots\}$ denote the set of items. We need notation for a *trade*; let $T_l \in \{-1, 0, 1\}^{|\mathcal{G}|}$ denote an *indicator vector* for a trade, such that agent l buys items $\{x \mid T_l(x) = 1, x \in \mathcal{G}\}$ and sells items $\{x \mid T_l(x) = -1, x \in \mathcal{G}\}$. Let $\mathbf{T} = (T_1, \dots, T_{|\mathcal{L}|})$ denote a complete trade between all agents.

Bids and asks define a *reported* value, $\hat{v}_l(T_l)$ for a trade T_l, comprising buys and sells. Bids indicate positive value for buying a bundle of items, while asks indicate negative value for selling a bundle of items. For example, if agent 1 submits bid $(AB, 10)$ and ask $(C, 5)$, then $\hat{v}_1([1, 1, 0]) = 10, \hat{v}_1([0, 0, -1]) = -5, \hat{v}_1([1, 1, -1]) = 5$. The values for other trades are constructed to be consistent with value $-\infty$ for selling anything other than item C, zero value for buying $S \subset \{AB\}$, and no additional value for buying more than bundle AB.

Let $\mathbf{T}^*$ denote the value-maximizing trade, given reported values, $\hat{v}_l(T_l)$, from each agent, with total surplus $V^* = \sum_l \hat{v}_l(T_l^*)$. Trades must be *feasible*, so that supply and demand is balanced, given a model of aggregation. Also, let $(V_{-l})^*$ denote surplus from the value-maximizing trade *without* bids (or asks) from agent l.

By definition, the Vickrey payment to agent l is computed as:

$$p_{\text{vick},l} = (V_{-l})^* - V_{-l}^*$$

where V_{-l}^* is the value of trade $\mathbf{T}^*$ to all agents except agent l, i.e. $V_{-l}^* = V^* - \hat{v}_l(T_l^*)$. Negative payments $p_{\text{vick},l} < 0$ indicate that the agent *receives* money from the exchange after it clears.

We can express an agent's Vickrey payment as a discount, $\Delta_{\text{vick},l}$, from the payment, $\hat{v}_l(T_l^*)$, the agent would make given its bid and ask prices; i.e. $p_{\text{vick},l} = \hat{v}_l(T_l^*) - \Delta_{\text{vick},l}$, where the *Vickrey discount* is computed as:

$$\Delta_{\text{vick},l} = V^* - (V_{-l})^*$$

The Vickrey discount is always non-negative, representing smaller payments by buyers and higher payments to sellers.

Economic Properties. Vickrey payments are IR, because $V^* \geq (V_{-l})^*$ by a simple feasibility argument, and also strategy-proof. The proof of strategy-proofness is omitted due to lack of space, but closely follows standard Vickrey proofs, for example see Varian & MacKie-Mason (1995). However, BB will often fail in an exchange, as we show in the next section.

Vickrey Budget-Balance: Success & Failure

Now that we have defined Vickrey payments in a combinatorial exchange, let us outline some cases in which BB is achieved and some cases in which BB fails. We will see that budget-balance failure is quite pervasive with Vickrey payments in exchanges.

Standard Exchange. First, consider a standard exchange with bids and asks for single units of a homogeneous item. In this case the exchange is cleared by sorting bids in order of decreasing price and asks in order of increasing price. Bids are matched with asks while the bid price is greater than the ask price. It is well known that Vickrey payments are not BB in this environment.

Let p^0_{bid} denote the smallest successful bid and p^{-1}_{bid} denote the largest unsuccessful bid. Similarly, let p^0_{ask} denote the largest successful ask and p^{-1}_{ask} denote the smallest unsuccessful ask. In the Vickrey scheme, every winning seller receives payment $p_{\text{vick},sell} = \min(p^0_{bid}, p^{-1}_{ask})$, whatever its own ask price, and every winning buyer pays $p_{\text{vick},buy} = \max(p^0_{ask}, p^{-1}_{bid})$, whatever its own bid price. The following condition is required for BB:

Claim 1. *Budget-balance is achieved in a simple exchange for homogeneous items and single-item bids and asks if and only if one (or more) of the following conditions hold: (1) $p^0_{bid} = p^0_{ask}$; (2) $p^0_{bid} = p^{-1}_{bid}$; (3) $p^0_{ask} = p^{-1}_{ask}$.*

Proof sketch. BB holds if and only if $\max(p^0_{ask}, p^{-1}_{bid}) \geq \min(p^0_{bid}, p^{-1}_{ask})$, leading to cases: (1) $p^0_{ask} \geq p^{-1}_{bid}$ and $p^0_{bid} \leq p^{-1}_{ask}$; (2) $p^0_{ask} < p^{-1}_{bid}$ and $p^0_{bid} \leq p^{-1}_{ask}$; (3) $p^0_{ask} \geq p^{-1}_{bid}$ and $p^0_{bid} > p^{-1}_{ask}$.

In other words, either one or more of the supply or demand curves must be "smooth" at the clearing point, with the first excluded bid at approximately the same bid price as the last accepted bid, *or* the winning bid and ask prices must precisely coincide. Thus, we cannot expect BB with Vickrey payments even in a standard (non-combinatorial) exchange except in special cases.

Combinatorial Exchange As an example of BB failure, consider that agents submit truthful bids in the earlier example; i.e. asks (A, \$10), (B, \$5) and bids (AB, \$51), (AB, \$40). $V^* = 51 - 10 - 5 = 36$, $(V_{-1})^* = (V_{-2})^* = 0$, $(V_{-3})^* = 25$, and $(V_{-4})^* = 36$. Agent 1's Vickrey payment is -10 - (36 - 0) = -46, agent 2's is -5 - (36 - 0) = -41, agent 3's is 51 - (36 - 25) = 40. The exchange runs at a loss of \$47 to the market maker.

One-Sided Vickrey-Payments First, a positive special-case. Claim 2 gives a sufficient condition for BB in the special-case that Vickrey discounts are only allocated to agents on one-side of an exchange, i.e. to all buyers or to all sellers (but not to buyers and sellers).

We define *aggregation* on the sell-side as when bids from multiple buyers can be combined to match an ask from a single seller, and aggregation on the buy-side as when asks from multiple sellers can be combined to match a bid from a single buyer.

Claim 2. *Budget-balance holds if Vickrey payments are implemented on one-side of an exchange, and when that side has no aggregation.*

Proof sketch. Simple, just show that this BB holds for each "cluster" of trading agents, and therefore for the entire exchange.

Bilateral matching is a special-case, with no aggregation on either side; i.e. Vickrey payments are budget-balanced if implemented for at most one agent in each trade, for example with trades cleared at either the ask price (buy-side strategy-proofness) or the bid price (sell-side strategy-proofness). Similarly, the single-item Vickrey auction is a special case (and strategy-proof to buyers but not the seller).

The Generalized Vickrey Auction (GVA) is the VCG mechanism for a combinatorial auction, in which there is a single seller and sell-side aggregation. The GVA is BB because the buyers, but not the seller, receive Vickrey payments. The auctioneer simply collects the total payment made by the buyers and passes it on to the seller. As such the GVA is strategy-proof for buyers but not for the seller. Another problem is that the seller can sometimes receive less than her ask price. Consider a seller with an ask price of (AB, \$10) and bids of (A, \$8) and (B, \$8) from different buyers. Each buyer receives Vickrey discount \$6 and pays \$2, but the seller needs at least \$10.

One-to-N models We can state a general negative result for Vickrey payments to all agents (buyers and sellers) in a combinatorial auction.

Claim 3. *Budget-balance fails with Vickrey payments to all agents in a combinatorial auction except in the case that no buyer requires a Vickrey discount.*

Proof sketch. Simple, just show that the seller extracts *all* of the surplus as its Vickrey discount.

Intuitively, BB fails in this case unless the marginal value contributed by each buyer is zero, i.e. unless the surplus in the exchange is the same with any one of the buyers removed.

Budget-Balanced Payment Rules

In this section we take BB and IR as hard constraints and propose methods to distribute surplus when an exchange clears to minimize the distance between discounts and Vickrey discounts. The choice of distance function has a distributional effect on the allocation of surplus and changes the incentive-compatibility properties of the exchange. In a later section we demonstrate useful truth-revelation properties for the Vickrey-based schemes.

We do the following:

- Formulate the pricing problem as a mathematical program, to minimize the distance to Vickrey payments with BB and IR as hard constraints.
- Introduce possible distance functions and construct corresponding budget-balanced payment schemes.
- Present a theoretical analysis of each payment scheme in a simple bidding model.

Mathematical Programming Model

We formulate the pricing problem as a linear program, to assign surplus to agents to minimize distance to Vickrey discounts. Let V^* denote the available surplus when the exchange clears, before any discounts, and $L^* \subseteq \mathcal{L}$ denote the set of agents that trade. Each agent may perform a number of buys and sells, depending on its bids and asks of other agents. We compute discounts $\Delta = (\Delta_1, \ldots, \Delta_L)$ to minimize the distance $\mathbf{L}(\Delta, \Delta_{\text{vick}})$ to Vickrey discounts, for a suitable distance function $\mathbf{L}$.

$$\min_{\Delta} \quad \mathbf{L}(\Delta, \Delta_{\text{vick}}) \qquad \text{[PP]}$$

$$\text{s.t.} \quad \sum_{l \in L^*} \Delta_l \leq V^* \qquad \text{(BB)}$$

$$\Delta_l \leq \Delta_{\text{vick},l} \quad , \forall\, l \in L^* \qquad \text{(VD)}$$

$$\Delta_l \geq 0 \quad , \forall\, l \in L^* \qquad \text{(IR)}$$

Constraint (BB) gives worst-case (or ex post) budget-balance, such that the exchange never makes a net payment to agents. We might also substitute an expected surplus $\overline{V^*}$ for V^* and implement average-case (or ex ante) budget-balance. Constraints (IR) ensure that truthful bids and asks are individual-rational for an agent, with a worst-case (or ex post) non-negative expected utility. Constraints (VD) ensure that no agent receives more than its Vickrey discount.[4]

In addition to the standard $\mathbf{L}_2$ and $\mathbf{L}_\infty$ distance metrics, we also consider the following functions: (a) $\mathbf{L}_{\text{RE}}(\Delta, \Delta_{\text{vick}}) = \sum_l \frac{\Delta_{\text{vick},l} - \Delta_l}{\Delta_{\text{vick},l}}$, a relative error function; (b) $\mathbf{L}_{\Pi}(\Delta, \Delta_{\text{vick}}) = \prod_l \frac{\Delta_{\text{vick},l}}{\Delta_l}$, a product error function; (c) $\mathbf{L}_{\text{RE2}}(\Delta, \Delta_{\text{vick}}) = \sum_l \frac{(\Delta_{\text{vick},l} - \Delta_l)^2}{\Delta_{\text{vick},l}}$, a squared relative error function; (d) $\mathbf{L}_{\text{WE}}(\Delta, \Delta_{\text{vick}}) = \sum_l \Delta_{\text{vick},l}(\Delta_{\text{vick},l} - \Delta_l)$, a weighted error function. The $\mathbf{L}_1$ metric provides no distributional information.

We drop agents with $\Delta_{\text{vick},l} = 0$ from all models, and simply set $\Delta_l = 0$ for these agents.

Payment Rules Rather than solving problem [PP] directly, we can compute an analytic expression for the family of solutions that correspond to each distance function. Each family of solutions is a parameterized *payment rule*. For example, the *Threshold* rule, $\Delta_l = \max(0, \Delta_{\text{vick},l} - C)$ for some parameter $C \geq 0$, solves [PP] for the $\mathbf{L}_2$ distance metric. For large C, Threshold allocates small, or no, discounts, while for $C = 0$ Threshold allocates Vickrey discounts.

To understand the construction of Threshold from $\mathbf{L}_2$ consider the simplest case, when constraints (VD) and (IR) are not binding, and perform Lagrangian optimization. Dropping the outer square root from the $\mathbf{L}_2$ metric and introducing Lagrange multiplier $\lambda \geq 0$, we have $\min \sum_l (\Delta_{\text{vick},l} - \Delta_l)^2 + \lambda(\sum_l \Delta_l - V^*)$. Now, computing first derivatives w.r.t. Δ_l and setting to zero, we have $-2(\Delta_{\text{vick},l} - \Delta_l) + \lambda = 0$ for all l.[5] Solving, this equates the distance to Vickrey discounts across all agents, $\Delta_{\text{vick},1} - \Delta_1 = \Delta_{\text{vick},2} - \Delta_2 = \ldots$, and with budget-balance we find $\Delta_l = \Delta_{\text{vick},l} - (\sum_{l'} \Delta_{\text{vick},l'} - V^*)/|L^*|$. This is the Threshold rule with parameter $C = (\sum_l \Delta_{\text{vick},l} - V^*)/|L^*| = \lambda/2$.

Distance Function	Payment Scheme	Discount Definition
$\mathbf{L}_2, \mathbf{L}_\infty$	Threshold	$\max(0, \Delta_{\text{vick},l} - C)$
$\mathbf{L}_{\text{RE}}$	Small	$\Delta_{\text{vick},l}$ if $\Delta_{\text{vick},l} \leq C$
$\mathbf{L}_{\text{RE2}}$	Fractional	$\mu \Delta_{\text{vick},l}$
$\mathbf{L}_{\text{WE}}$	Large	$\Delta_{\text{vick},l}$ if $\Delta_{\text{vick},l} \geq C$
$\mathbf{L}_{\Pi}$	Reverse	$\min(\Delta_{\text{vick},l}, C)$
-	No-Discount	0
-	Equal	D

Table 1: Distance Functions and Payment Schemes.

Rule	Vick	Equal	Frac	Thresh	Reverse	Large		Small	
Agent 1	-46	-22	-25.6	-28	-22.5	-46 or -10		-35 or -10	
Agent 2	-41	-17	-20.6	-23	-17.5	-5	-41	-5	-30
Agent 3	40	39	46.2	51	40	51	51	40	40

Table 2: Payments with Different Rules in the Simple Problem.

Table 1 tabulates the payment rules for each distance function, and also includes the *Equal* rule which is not Vickrey-based but divides surplus equally across all agents, and the *No-Discount* rule (see also Figure 1). Each payment rule is parameterized with $C \geq 0$, except for *Fractional*, which has parameter $0 \leq \mu \leq 1$. The parameters that give BB in each scheme can be easily computed from Vickrey discounts and available surplus.

Example. In Table 2 we compare the payments made with each payment scheme in our simple problem. Notice that neither the Large or Small schemes provide useful guidance about how to distribute the discount across the two sellers, this depends on how the tie is broken.

Theoretical Analysis

In this section we develop simple analytic results for the amount of manipulation an agent will perform with each payment scheme. The model permits tractable analysis, and proves interesting both for the insight it provides and for the close correspondence that we find with later experimental results for combinatorial exchange problems.

We choose to analyze an exchange in which bids and asks are for *single items*. Later, in our experimental analysis we compare the payment schemes in combinatorial problems.

For buyers (the analysis is symmetric for sellers):
(1) Every agent l has value v_l for a single item (drawn from some distribution $F_l(v)$) and chooses to manipulate by

[4] The (VD) constraints are not redundant for certain distance metrics, such as the $\mathbf{L}_{\text{RE}}(\cdot)$ metric.

[5] First-order conditions are necessary and sufficient for optimality in this problem because the Hessian is positive definite.

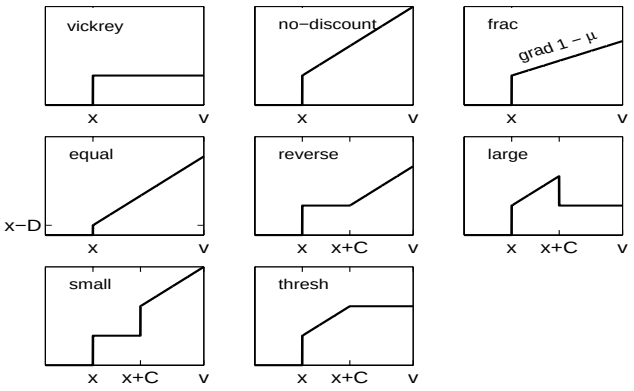


Figure 1: Bid price b_l (x-axis) against adjusted bid price $b_l - \Delta_l$ (y-axis) in each payment scheme. Agent value v, highest outside bid x, Payment scheme parameters C, μ, D.

$\theta_l \geq 0$, and bid $b_l = v_l - \theta_l$.
(2) The maximum bid x_l from another agent for the item, or ask price (whichever is higher), is uniformly distributed about v_l, i.e. $x_l \sim U(v_l - \delta, v_l + \delta)$ for some constant, $\delta > 0$.
(3) The average surplus available to the market maker when the exchange clears is $\alpha\delta$ per-agent, for some constant $\alpha > 0$ that defines the amount of surplus.
(4) In equilibrium, the market maker selects a parameter (e.g. C) for the payment scheme to achieve average-case budget-balance. Payment rules are computed *before* agents bid, and the parameters are known to bidding agents.

Agent l has a quasi-linear utility function, $u_l = v_l - p$, for submitting the highest bid where p is its payment to the exchange, i.e. $p = b_l - \Delta_l$. Figure 1 illustrates each payment rule in this simple model, plotting bid price b_l against adjusted price $b_l - \Delta_l$; e.g., in Vickrey the agent pays only x for any bid $b_l \geq x$, in Threshold the agent pays $b_l = x + C$ for $b_l \geq x + C$, and b_l for $x \leq b_l < x + C$, given parameter C, etc.

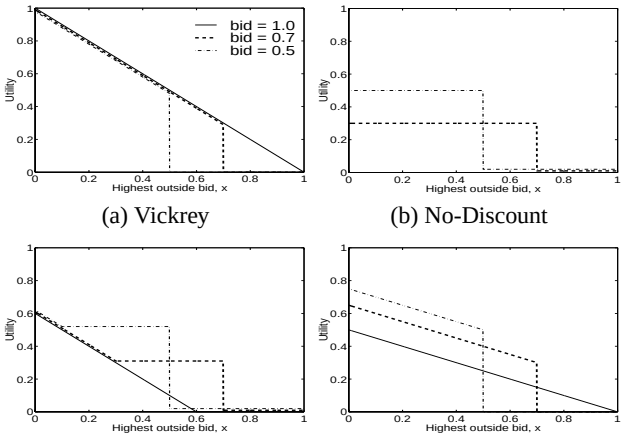
For each payment scheme we determine: (a) an agent's optimal bidding strategy as a function of the parameters of the rule, e.g. C or μ; and (b) the equilibrium parameterization of the rule, e.g. value for C, that leads to budget-balance given that agents follow this optimal bidding strategy. The analysis leads to a relationship between the *available surplus and the degree of manipulation* for each payment rule (see Figure 4).

One can be critical of our assumptions. We leave undefined both the valuation distribution functions $F_l(v)$ and the distribution that defines the item an individual agent values. It is quite likely that there are no $F_l(v)$ that are consistent with our assumption of a uniformly distributed second-highest bid in equilibrium. In addition, we adopt average-case budget-balance and compute payment rules before agents bid, but ignore any effect that rules have on surplus via agents' bidding strategies.

However, we believe that this analysis has significant value. Its main success is that it clearly demonstrates the effect that different types of budget-balanced Vickrey-based payment rules can have on agent manipulation. We leave a full equilibrium analysis for future work.

Graphical Intuition. Manipulation has two effects on the expected utility for an agent: (i) the probability of the adjusted bid being accepted decreases, and (ii) the total utility if the bid is accepted can go up because the agent's payment might be reduced. Payment rules change (ii) but not (i), and in turn effect agents' bids and the efficiency of the exchange.

In Figure 2 we plot the *utility* for a particular bid, $b = v - \theta$, as the value of the outside bid x varies, for payment schemes Vickrey, No-Discount, Threshold and Fractional. Each subplot is for a single scheme, with individual curves corresponding to different bids.[6]



(a) Vickrey (b) No-Discount

(c) Threshold (d) Fractional

Figure 2: Utility of bids $b = v - \theta$ with $\theta = \{0, 0.3, 0.5\}$, $v = 1$, as the best outside bid x varies between 0 and 1. $C = 0.4$ in Threshold, and $\mu = 0.5$ in Fractional.

In the Vickrey scheme a lower bid reduces the agent's expected utility because it decreases the probability of success without increasing the utility of a successful bid. In comparison, with no discount the agent gains utility on all successful bids by the amount of deviation from truthful bidding. In the Threshold scheme a lower bid only reduces the price paid for a limited range of outside bids (closer than C to the bid price), while in the Fractional scheme a lower bid reduces the price paid on all successful bids (but by less than in the No Discount scheme).

Making our assumption about the distribution of x around an agent's value v_l, we can compute the expected utility for different levels of manipulation under each scheme as the area under a particular curve in a plot like Figure 2.[7] The expected-utility maximizing bid corresponds to the curve with maximum area. In Figure 3 we plot the expected *gain* in utility (in comparison with truthful bidding), $\mathrm{E}u(\theta) - \mathrm{E}u(0)$, for bid $b = v - \theta$ in each payment rule. Rule parameters are set to give BB with surplus $\alpha = 0.1$ at optimal agent strate-

[6] Although not plotted here, the curves for Equal are similar to the No-Discount case (except shifted higher in utility by a constant amount), and Large is similar to Threshold.

[7] It is at this stage that an equilibrium analysis would need to use a derived expression for the distribution of x.

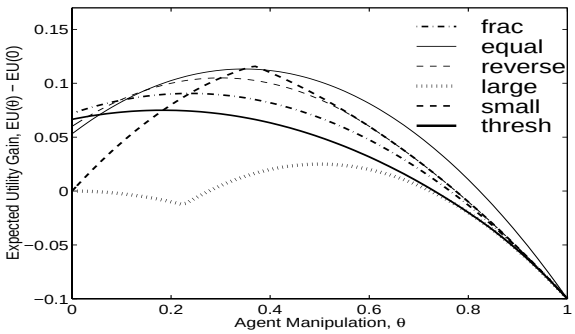


Figure 3: Expected Gain in Utility for different bids $b = v - \theta$ under each payment scheme, with rule parameters set to give BB with surplus $\alpha = 0.1$.

gies. Notice that the level of manipulation, θ^*, that maximizes the agent's gain in utility is smallest in the Threshold scheme for this value of surplus.

The results (below) show that the Large and Threshold rules perform well in this model, and lead to the following intuitive remarks about payment rules (see Figure 1):
1. A large flat section for bids close to the agent's true value is useful, i.e. with adjusted bid price independent of the agent's bid price.
2. Nowhere should the adjusted bid price be greater than the agent's bid price (for IR with truthful bidding), which constrains the line to lie to the right of the "no-discount" line.
3. It is more important to implement the flat section for values, v, far from the highest outside bid, x, than values close to the highest outside bid (i.e. *Large* rather than *Small*), because manipulation is already more risky for true values close to v than far from x.[8]

It is useful to think about the "degree of manipulation freedom" available to an agent, which in this simple single-bundle model is the difference between an agent's value v and the highest outside bid x. In general, this is simply measured by the Vickrey discount to an agent that bids truthfully, i.e. the amount by which it could have reduced its bid price and still participated in the same trades. The Large and Threshold schemes are effective because they make manipulation more difficult and less useful for an agent with a large degree of manipulation freedom, while leaving the ability to manipulate of agents with a small degree of manipulation freedom unchanged. This is a good incentive strategy because it attacks the "low risk" manipulation opportunities, but leaves the "high risk" opportunities. Agents are uncertain about the bids from other agents and always run the risk of bidding too low and forfeiting a profitable trade.

Results. Table 3 summarizes the analytical results, giving an agent's optimal bidding strategy, θ^*, as a function of the parameter in each scheme, and the expected discount per-agent given that optimal strategy.[9] We present an example derivation, for the Threshold rule, below.

[8]Note that in terms of efficiency the picture is mixed. While we can stand more manipulation from agents with large values compared to x, without changing the trades that we implement, if the bids from those agents does change the final implementation the effect on efficiency is likely to be quite large.

Rule	Optimal Manipulation, θ^*	Expected Discount
No-Discount	$\delta/2$	0
Vickrey	0	$\delta/4$
Fractional	$\max\left[0, \left(\frac{1-\mu}{2-\mu}\right)\delta\right]$	$\min\left[\delta/4, \frac{\delta\mu}{4(2-\mu)^2}\right]$
Threshold	$\min[C, \delta/2]$	$\max\left[0, \frac{(\delta-2C)^2}{4\delta}\right]$
Equal	$\frac{\delta-D}{2}$	$\frac{D(\delta+D)}{4\delta}$
Small	$\max[0, \min(\delta/2, \delta - C)]$	$\min\left[\delta/4, C^2/4\delta\right]$
Large	0, if $C \leq \delta/\sqrt{2}$ $\delta/2$, otherwise	$-C^2/4\delta + \delta/4$, if $C \leq \delta/\sqrt{2}$ 0, otherwise
Reverse	$\max\left[0, \frac{\delta-C}{2}\right]$	$\min[\delta/4, C/4]$

Table 3: Analytical results.

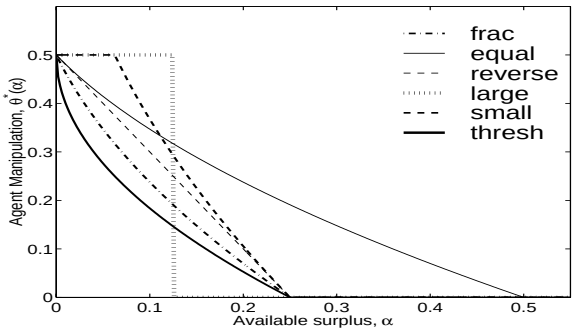


Figure 4: Optimal agent manipulation, $\theta^*(\alpha)$, (as a proportion of δ) under each payment scheme as the amount of available surplus increases from 0 to $\delta/2$ per-agent.

In Figure 4 we enforce BB, computing parameters in the payment schemes to set the expected discount equal to surplus $\alpha\delta$, and plot the equilibrium manipulation performed in each payment scheme as the amount of surplus varies. The Vickrey payment scheme can be implemented with surplus $\delta/4$ per-agent, so all schemes except Equal and No-Discount prevent manipulation completely for $\alpha \geq 0.25$. For smaller amounts of surplus the market maker is forced to deviate from Vickrey, and move left in Figure 4. At $\alpha = 0$ no schemes can provide any discount, and the agent manipulates by $\delta/2$.

First, notice that the simple minded Equal scheme appears to have bad incentive properties. In fact, the Threshold method dominates all other schemes in this model except Large. Large has an interesting bad-good phase transition at $\alpha = 1/8$, and can prevent manipulation completely for $1/8 \leq \alpha \leq 1/4$ even though agents with small Vickrey discounts might have benefited from manipulation with hindsight. Agent uncertainty coupled with the risk of bidding too low and either falling from the flat section or under-bidding the highest outside bid lead agents to bid truthfully.

[9]It is useful to confirm that all expressions reduce to that for the Vickrey and No-Discount rules at extreme parameter values (e.g. $\mu = \{0, 1\}$ in Fractional, $C = \{0, \delta/2\}$ in Threshold, etc.)

Example Derivation: Threshold Rule. Each agent receives discount $\max(0, b - (x + C))$, for some constant $C > 0$. The agent's utility given bid $v - \theta$, value v, highest outside bid x, and Threshold C, is computed as:

$$u(v\text{–}\theta, v, x, C) = \begin{cases} v\text{–}(x + C), & \text{if } v\text{–}\theta \geq x + C \\ v - (v\text{–}\theta), & \text{if } x + C > v\text{–}\theta \geq x \\ 0, & \text{otherwise} \end{cases}$$

Assume that $C \leq \delta$, so that the agent will receive a discount for some choice of $\theta < \delta$. Consider three cases.
Case (1), $\theta \leq \delta\text{–}C$. The expected utility for bid θ given C, $\mathrm{EU}(\theta, C)$ is:

$$\int_{x=v-\delta}^{v-\theta-C} [v\text{–}(x + C)] f(x)\mathrm{d}x + \int_{x=v-\theta-C}^{v-\theta} [v\text{–}(v\text{–}\theta)] f(x)\mathrm{d}x + \int_{x=v-\theta}^{v+\delta} 0 f(x)\mathrm{d}x$$
$$= \frac{(\delta\text{–}\theta\text{–}C)}{2\delta}(v\text{–}C)\text{–}\frac{1}{4\delta}\left[(v\text{–}\theta\text{–}C)^2\text{–}(v\text{–}\delta)^2\right] + \frac{\theta}{2\delta}(\theta + C\text{–}\theta)$$

In *case (2),* $\delta\text{–}C < \theta \leq \delta$, $\mathrm{EU}(\theta, C) = \int_{x=v-\delta}^{v-\theta} [v\text{–}(v\text{–}\theta)] f(x)\mathrm{d}x = \theta(\delta - \theta)/2\delta$. In *Case (3),* $\delta < \theta$, then $\mathrm{EU}(\theta, C) = 0$. ¿From this, the agent's optimal bidding strategy, denoted $\theta^*(C)$, by differentiation w.r.t. θ and case analysis, is:

$$\theta^*(C) = \min[C, \delta/2]$$

The discount to the agent for bid $b = v\text{–}\theta$ is: $\Delta(v - \theta, v, x, C) = \max[0, v\text{–}\theta\text{–}(x + C)]$. The expected discount, first in the case that $\theta \leq \delta - C$, is:

$$\mathrm{E}\Delta(\theta, C) = \int_{x=v-\delta}^{v-\theta-C} [v - \theta\text{–}(x + C)] f(x)\mathrm{d}x$$
$$= (v - \theta - C)\frac{(\delta - \theta - C)}{2\delta} - \frac{1}{4\delta}\left[(v - \theta - C)^2\text{–}(v - \delta)^2\right]$$

or $\mathrm{E}\Delta(\theta, C) = 0$ in the case $\theta > \delta\text{–}C$. Substituting for the agent's optimal bidding strategy $\theta^*(C)$ we have: $\mathrm{E}\Delta(\theta^*(C), C) = \max\left[0, \frac{(\delta - 2C)^2}{4\delta}\right]$. Now, with per-agent surplus $\alpha\delta$ and budget-balance, such that $\mathrm{E}\Delta^*(C) \leq \alpha\delta$, the exchange should set $C^*(\alpha) = \min\left[0, \frac{\delta}{4}\left(2 - \sqrt{16\alpha}\,\right)\right]$ to minimize manipulation.

Experimental Analysis

In this section we provide an experimental analysis of the payment schemes in a set of combinatorial problem instances. Agents are either buyers or sellers, and values are assigned to agents for *bundles* following the Random, Weighted Random, Decay, and Uniform distributions from Sandholm (1999), adapted in this case to a combinatorial exchange. Each agent submits bids (asks) for multiple bundles, with exclusive-or constraints across bids (asks). We test problems with 5, 10, and 20 agents, a total of 100 bids and asks (evenly distributed across agents), 50 goods, and with different proportions of buyers and sellers.[10]

[10]Results are averaged over 80 problem instances, for numbers of Buyer/Sellers $\in \{5/5, 7/3, 2/3, 4/1, 10/10, 15/5\}$.

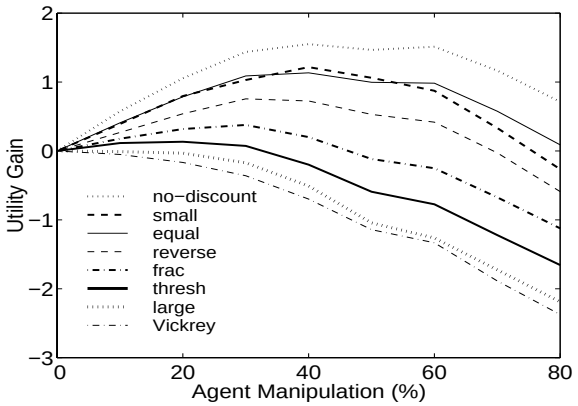


Figure 5: Average Single-Agent Gain in Utility from manipulation by y% (vs. truthful bidding), in a system in which every other agent manipulates by y%. Problem size: 5 buyers/5 sellers.

In our theoretical model we adopted average-case budget-balance to make the analysis tractable. We now revert to the more natural worst-case (or every-time) budget-balance in which the market maker distributes exactly the available surplus every time the exchange is cleared. Payment rules are now computed *after* bids are received.

We perform a limited strategic analysis. First, we assume that the strategy of agent l is to adjust *all* its bids and asks by the same fractional amount, y_l %, i.e. submitting bid prices y_l% below value and ask prices y_l% above value. Second, we look for a *symmetric* Nash equilibrium in which every agent follows the same strategy, for some y%. Finally, we compute an *approximation* to this equilibrium for computational tractability. We compute the average utility gain to a single agent for 0% vs. y% manipulation, given that every other agent manipulates by y%, and determine the amount of manipulation, y^*, that maximizes this utility gain. We assume that this is also the optimal strategy for an individual agent against a population of agents with *fixed* strategies y^*, and therefore the Nash equilibrium.[11]

Given this, we read off the symmetric Nash equilibrium under a particular payment rule as the peak of a plot such as that in Figure 5, which plots the gain in utility for strategy y% vs. 0% in a system in which every agent follows strategy y%, in this case for the 5 buyers/5 sellers problem set. In this case, notice that the equilibrium manipulation level in Large and Threshold is less than under the other rules, in this case around 10% and 20% in Large and Threshold, compared with 30%, 40% and 50% in Fractional, Equal and No-Discount. In addition, the amount of utility gain in Large and Threshold is much less than in the other schemes.

In Table 4 we summarize the results of experiments across all problem sets. We compare: the average utility gain, and the correlation with Vickrey discounts, at manipulation levels of 10%, 20% and 30% in each scheme; and the average optimal degree of manipulation by agents in each scheme,

[11]One benefit of this technique is that we have a method to measure the degree of manipulation even when there is in fact *no* symmetric pure Nash equilibrium.

	No-Discount	Vickrey	Small	Frac
Utility Gain	0.799	-0.195	0.479	0.211
Correlation	0.053	1.0	0.356	0.590
Manipulation, θ^*	48	0	48	32
Efficiency (%)	58	100	58	78
	Threshold	Equal	Large	Reverse
Utility Gain	0.110	0.516	0.029	0.337
Correlation	0.543	0.356	0.176	0.522
Manipulation, θ^*	22	46	18	44
Efficiency (%)	86	62	88	64

Table 4: Experimental results. Utility gain and Correlation with Vickrey discounts computed for manip. 10%, 20% and 30%, and averaged over all problem instances (for 5–20 agents).

and the corresponding allocative efficiency. The allocative efficiency in the Large and Threshold schemes is considerably higher than in the other schemes.

Discussion

The partial ordering $\{\text{Large, Threshold}\} \succ \text{Fractional} \succ \text{Reverse} \succ \{\text{Equal, Small}\}$ from the experimental results is remarkably consistent with the results of our theoretical analysis. Although the Large scheme generates slightly less manipulation and higher allocative efficiency than Threshold, the correlation between discounts and Vickrey discounts is much greater in Threshold than Large. An agent's discount in Large is very sensitive to its bid, and we expect Large to be less robust than Threshold in practice because of this all-or-nothing characteristic.

As discussed earlier, we have made a number of assumptions, both in the analytic models of agent manipulation and also in the manipulation structure considered experimentally. In addition to understanding the effects of these assumptions, in future work we would also like to: quantify worst-case and average-case utility gains from manipulation in each payment scheme, given a particular amount of surplus; and derive *optimal* payment schemes, for example minimizing worst-case gains from manipulation. One avenue is to ask how bad would the efficiency get if every agent was perfectly informed about the other agents, and followed a best-possible bidding strategy given the payment rules. Finally, we suspect that stochastic payment rules might prove to have interesting incentive properties.

Conclusions

We constructed budget-balanced payment schemes to minimize different distance functions to Vickrey payments, and showed analytically and experimentally that a simple Threshold rule has better incentive properties than other payment schemes. The effect of the payment scheme is to implement a distribution of manipulation-preventing discounts across a population of agents to exploit an agent's inherent uncertainty about bids from other agents and the degree to which manipulation can be useful. The Threshold rule increases the amount by which an agent with a large degree of manipulation freedom must adjust its bid to have a useful effect on the price it finally pays, while leaving unaffected the manipulation properties for agents with a small degree of manipulation freedom.

Finally, we note that the schemes outlined here can also allow a market maker to make a small profit by taking a sliver of budget-balance, or used in combination with a participation charge to move payments closer to Vickrey payments.

Acknowledgments

We thank William Walsh, and the anonymous reviewers, for their helpful comments and suggestions. This research was funded in part by National Science Foundation Grant SBR 97-08965. The first author gratefully acknowledges financial support from an IBM Research Fellowship.

References

Barbera, S., and Jackson, M. O. 1995. Strategy-proof exchange. *Econometrica* 63(1):51–87.

Clarke, E. H. 1971. Multipart pricing of public goods. *Public Choice* 11:17–33.

de Vries, S., and Vohra, R. 2000. Combinatorial auctions: A brief survey. Tech. report, MEDS Department, Kellogg Graduate School of Management, Northwestern University.

Groves, T. 1973. Incentives in teams. *Econometrica* 41:617–631.

Kalagnanam, J. R.; Davenport, A. J.; and Lee, H. S. 2000. Computational aspects of clearing continous double auctions with assignment constraints and indivisible demand. IBM Research Report RC 21660 (97613). To appear in *Electronic Commerce Research Journal.*

McAfee, R. P. 1992. A dominant strategy double auction. *J. of Economic Theory* 56:434–450.

Myerson, R. B., and Satterthwaite, M. A. 1983. Efficient mechanisms for bilateral trading. *Journal of Economic Theory* 28:265–281.

Rothkopf, M. H.; Pekeč, A.; and Harstad, R. M. 1998. Computationally manageable combinatorial auctions. *Management Science* 44(8):1131–1147.

Sandholm, T. 1999. An algorithm for optimal winner determination in combinatorial auctions. In *Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 542–547.

Varian, H., and MacKie-Mason, J. K. 1995. Generalized Vickrey auctions. Tech. report, University of Michigan.

Vickrey, W. 1961. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance* 16:8–37.

Walsh, W.; Wellman, M.; and Ygge, F. 2000. Combinatorial auctions for supply chain formation. In *Proc. ACM Conference on Electronic Commerce*, 260–269.

Wurman, P. R.; Walsh, W. E.; and Wellman, M. P. 1998. Flexible double auctions for electronic commerce: Theory and implementation. *Decision Support Systems* 24:17–27.

Agent-Human Interactions in the Continuous Double Auction

Rajarshi Das, James E. Hanson, Jeffrey O. Kephart and Gerald Tesauro
Institute for Advanced Commerce
IBM T.J. Watson Research Center
30 Saw Mill River Road, Hawthorne, NY 10532, USA

Abstract

The Continuous Double Auction (CDA) is the dominant market institution for real-world trading of equities, commodities, derivatives, etc. We describe a series of laboratory experiments that, for the first time, allow human subjects to interact with software bidding agents in a CDA. Our bidding agents use strategies based on extensions of the Gjerstad-Dickhaut and Zero-Intelligence-Plus algorithms. We find that agents consistently obtain significantly larger gains from trade than their human counterparts. This was unexpected because both humans and agents have approached theoretically perfect efficiency in prior all-human or all-agent CDA experiments. Another unexpected finding is persistent far-from-equilibrium trading, in sharp contrast to the robust convergence observed in previous all-human or all-agent experiments. We consider possible explanations for our empirical findings, and speculate on the implications for future agent-human interactions in electronic markets.

1 Introduction

We envision a future in which economically intelligent and economically motivated software agents will play an essential role in electronic commerce. Among the present-day glimmerings of such a future are simple bidding agents offered by auction sites such as eBay and Amazon and by third-party bidding services such as eSnipe[1], pricebots such as buy.com that automatically undercut the competition, and shopbots such as DealTime that minimize the total cost of a bundle of goods by partitioning it across one or more vendors, taking shipping cost schedules into account. It is natural to expect continued growth in the variety and sophistication of automated economic decision-making technologies such as these. In addition, we anticipate the emergence of an even larger and more diverse class of agents for which economic decision-making capabilities are still essential, but ancillary to the primary function of providing information goods and services to humans or other agents [Kephart *et al.*, 2000]. (Throughout this paper we use the term "agent" to refer exclusively to a software agent, as opposed to a human economic agent.) Whether their main business is ontology translation, matchmaking, network service provision, or anything else, these agents will charge a fee for their goods or services, and will negotiate both as buyers and as sellers with other agents. Thus they will have to be economically intelligent, capable of making effective decisions about pricing, purchasing, or bidding.

If this vision is to be realized, then it must be demonstrated that, within their domain of application, agents can attain a level of economic performance that rivals or exceeds that of humans on average, without introducing undue risk. Otherwise, people would not entrust agents with making economic decisions.

The purpose of this paper is to provide such a demonstration. Through a series of controlled laboratory experiments in which humans and agents participate simultaneously in a realistic auction (a Continuous Double Auction, or CDA), we show that software agents can consistently obtain greater gains from trade than their human counterparts. In a sense, this work can be viewed as another chapter in the venerable AI tradition of human vs. machine challenges. Already, machine supremacy has been demonstrated in two-player games such as backgammon, checkers, and chess, and a serious attack is now being made on games such as bridge and poker [Schaeffer, 2000], in which there are slightly more than two players who play in a well-defined sequence. In contrast, the number of players in the CDA is typically much greater (we limit it to 12 in this report), and the moves by individual players are completely independent and asynchronous. These and other features make game-theoretic analysis of the CDA intractable. Another notable difference is that the successful demonstration of machine superiority in the CDA and other common auctions could have a much more direct and powerful financial impact—one that might be measured in billions of dollars annually.

This paper is organized as follows. Section 2 defines the CDA and discusses the relationship of our work to previous studies by economists and computer scientists. Section 3 provides a brief overview of the technological infrastructure used in the experiments. Section 4 describes the agent environment and the individual agent strategies. Section 5 provides details of the market rules and experimental parameters, and

[1] eSnipe automates a common practice among eBay bidders of waiting until a few seconds before an auction's close to place a bid.

section 6 presents the results for two different agent strategies. We conclude with a brief summary and discussion of implications and future directions in section 7.

2 Background on the CDA

Our laboratory study of economic interactions between agents and humans utilizes a simplified model of a Continuous Double Auction (CDA) market. The CDA is one of the most common exchange institutions, and is in fact the primary institution for trading of equities, commodities and derivatives in markets such as NASDAQ and the NYSE. In the CDA, there is a fixed-duration trading period, and buy orders ("bids") and sell orders ("asks") may be submitted at any point during the period. If at any time there are open bids and asks that are compatible in terms of price and quantity of good, a trade is executed immediately. Typically, an announcement is broadcast immediately to all participants when orders are placed or trades are executed.

In our model CDA, multiple units of a single hypothetical commodity can be bought or sold. Participants are assigned a fixed role of either Buyer (only submits bids) or Seller (only submits asks). There are several periods of trading; at the start of each period, participants are given a list of "limit prices" (values for Buyers and costs for Sellers) for each unit to be bought or sold. The limit prices are held constant for several periods and periodically shifted by random amounts to test responsiveness to changing market conditions. Each participant's objective is to maximize "surplus," defined as (limit price - trade price) for buyers and (trade price - limit price) for sellers.

The assumptions of fixed roles and fixed limit prices conform to extensive prior studies of the CDA, including experiments involving human subjects [Smith, 1962; 1982] and automated bidding agents [Cliff and Bruten, 1997; Gjerstad and Dickhaut, 1998]. Under such assumptions, a market consisting of rational players will eventually converge to steady trading at an equilibrium price p^*, at which there is a balance between Supply (the total number of units that can be sold for positive surplus) and Demand (the total that can be bought for positive surplus). For each participant, one can define a theoretical surplus as the total surplus that would be obtained if all units traded at p^*. One can also define a participant's efficiency μ as the ratio of actual surplus to theoretical surplus. In human subject studies [Smith, 1962; 1982], convergence close to equilibrium was found within several periods, with the approach towards p^* exhibiting a "scalloped" shape (i.e., a decelerating curved trajectory) of progressively smaller amplitude in each successive period. Robust convergence to equilibrium was also found in homogeneous populations of agents [Cliff and Bruten, 1997; Gjerstad and Dickhaut, 1998], with smaller-amplitude scallops than in the all-human experiments. Both all-human and all-agent CDA studies claimed very high population efficiency, ranging between 0.95 and 1.0.

Our work differs from prior CDA studies in two significant ways. First, we are interested in studying and understanding interactions between agent and human bidding strategies. Second, we focus primarily on measuring and understanding the performance of individual agents, instead of global measures of aggregate market behavior. As agent designers, we would like to understand the principles by which robust bidding strategies can be designed that perform well against both human and computerized opposition. Our focus on competition in heterogeneous bidder populations is similar to that of the agent vs. agent competition held at the Santa Fe Double Auction Tournament (SFDAT) [Rust *et al.*, 1992]. The SFDAT was an intrinsically discrete-time auction, with non-persistent orders, synchronized bidding of all agents at every time step, and a coarse time step size deliberately chosen to allow all agents enough time to calculate and place their bids. Thus the conclusions may not apply to trading in normal real-time CDA markets. Another market-based tournament for bidding agents, the Trading Agent Competition (TAC) [Wellman *et al.*, 2001] was held in conjunction with ICMAS-00. This competition was much more realistic in design, and required the agents to simultaneously participate in multiple markets, each of which required a different bidding strategy. Our study incorporates a degree of realism in market dynamics and messaging/communication similar to that of TAC, while preserving a classical CDA design. This allows both agents and humans to participate, as well as facilitating comparisons with prior all-human and all-agent CDA studies.

3 Overview of the Experiments

For our experiments with humans and agents, we developed a hybrid system that combined GEM, a special-purpose distributed system for experimental economics developed by members of the IBM Watson Experimental Economics Laboratory (WEEL), with Magenta, a prototype agent environment developed at IBM Research. The hybrid configuration is illustrated in Figure 1.

A real-time, asynchronous CDA was administered by a GEM auctioneer process running on a Windows NT workstation, which communicated with all bidders and executed trades when appropriate. To ensure that agents and humans could interact seamlessly with one another, and that there would be no subtle bias in their treatment, humans and agents used the same set of messages to communicate with the GEM auctioneer. Each human bidder was given a Windows NT workstation running a GEM client process that interpreted messages from the GEM auctioneer, and encoded messages to be sent back to it. This GEM client offered a GUI that permitted its user to view the order queue, the trade history, and his/her assigned parameters. It also permitted the user to enter bids or asks. Each Magenta bidder agent participated in the auction through a modified version of the GEM client that forwarded messages via TCP/IP to a UNIX workstation running the Magenta environment and the agents themselves. Each agent received messages forwarded to it by its modified GEM client. Whenever it wished, the agent could send messages to its GEM client, which forwarded them as quickly as possible to the GEM auctioneer. Thus the Magenta agents and the human bidders had access to identical streams of data from the auctioneer, and the auctioneer could not distinguish orders placed by humans from those placed by agents.

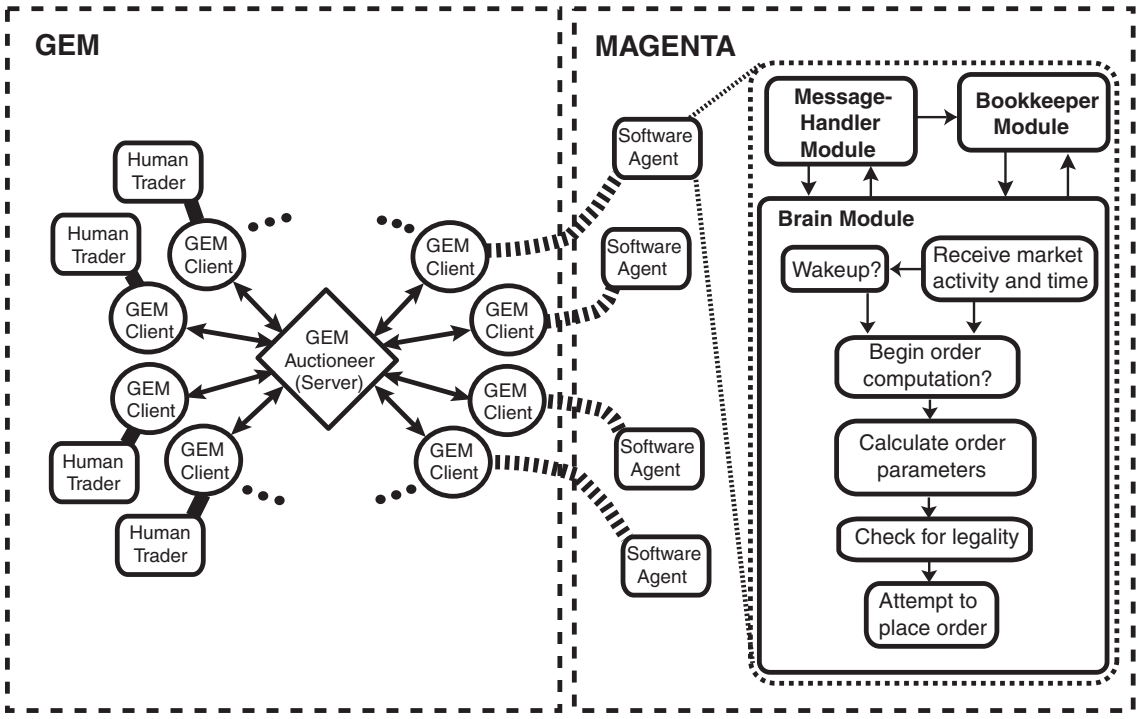


Figure 1: Hybrid GEM-Magenta configuration. At the left is the GEM system, showing a GEM auctioneer communicating with a set of GEM clients. Each GEM client either presents a GUI to a human trader or communicates via TCP/IP with one of the Magenta bidding agents shown at right. Also at right is an expanded view of an agent's architecture, showing the three internal modules described in the text and sketching the control flow within the Brain module.

4 Agents for the CDA

4.1 Agent environment and architecture

The bidder agents were implemented on top of Magenta, which provides messaging, naming, and directory services, and supports both one-shot messaging and conversations (correlated sequences of messages). Each bidder agent contained a MessageHandler module, a Bookkeeper module, and a Brain module, all of which were specifically tailored for the CDA.

The MessageHandler was responsible for formatting and sending outgoing messages, and for receiving and parsing incoming messages. Incoming messages included initialization information, notifications of bids and asks placed by other agents, of trades, and of time remaining in an auction period. Upon receipt of a message, the MessageHandler would parse it and take an appropriate action, such as handing the record of the bid, ask, or trade to the Bookkeeper module.

The Bookkeeper maintained the agent's internal representation of the market state (e.g., the history of orders and trades, current open orders, time remaining in the period, etc.) and of the agent's internal state (e.g., orders the agent had placed, the agent's inventory and available funds, etc.).

The Brain module was responsible for placing bids and asks. Each Brain placed orders on its own thread of execution, with its own activation schedule. Each Brain contained a module for a bidding strategy that determined order parameters based on the information stored in the Bookkeeper, and a timing module that governed the circumstances under which the brain's execution thread would wake up, apply the bidding strategy and (possibly) place an order. Outgoing order messages were formatted and sent back to the GEM client by the MessageHandler. Optionally, the timing module would awaken the brain's execution thread whenever a trade occurred and/or when a particularly attractive bid or ask was placed by another player.

4.2 Agent strategies

The Magenta bidding agents face the following task in a live auction: given the time remaining T in the trading period, a number N of tradeable units, a vector $\vec{L}$ of N limit prices (one for each unit), and the history H of previous activity in the market, calculate an order time τ and a price p.

The timing of orders was governed by a simple heuristic based on a sleep-wake cycle. The sleep time was set to a fixed interval of s seconds, with small random jitter $\pm(0-25)\%$ added. In our experiments, "fast" agents were defined by setting $s = 1$ and by permitting wakeup on all orders and trades. "Slow" agents were defined by setting $s = 5$ and allowing wakeups only on trades. When an agent wakes up, it computes an order price using its pricing algorithm. The agent will then submit the order, provided that the computed price is compatible with the market rules and its understanding of the current market conditions. If the agent currently has an open order, the order will be a replacement order; otherwise it will be a new one.

The pricing algorithms employed in these experiments are based on algorithms originally introduced for simpler ver-

sions of the CDA that lacked a persistent order queue, and made assumptions about market dynamics that are inconsistent with the notion of real-time, independent agents. We now describe modifications that we made to these algorithms to tailor them to our version of the CDA.

Zero-Intelligence-Plus (ZIP) Strategy

Cliff [Cliff and Bruten, 1997] proposed an algorithm called "Zero-Intelligence-Plus" (ZIP) to explore the minimum degree of agent intelligence required to reach market equilibrium in a simple version of the CDA. The market dynamics studied in [Cliff and Bruten, 1997] were unrealistic in that there was no explicit notion of time, no definite period length, and no persistent orders: submitted orders were either traded or removed instantaneously. We have modified ZIP to function in a real-time market with a definite period length and persistent open orders. The primary modification has to do with outbidding or undercutting existing orders. This now happens when orders remain open for a certain amount of real time without being traded. Our modifications turn out to be related to those independently proposed by Preist & van Tol [Preist and van Tol, 1998].

In our ZIP implementation, each agent maintains a vector of internal price variables $\vec{p}$; the i-th component of $\vec{p}$, p_i, is used to set the order price when trading the i-th unit. At the start of trading, $\vec{p}$ is initialized to random positive-surplus values, and is adjusted during the period according to the observed trading action.

When a trade occurs at trade price p_T, each p_i is adjusted by a small random increment in the direction of p_T. If the adjustment is in the direction of increasing profit margin (i.e. raising p_i for sellers and lowering p_i for buyers), the change is always made regardless of whether or not the i-th unit has already been traded. However, for adjustments in the direction of decreasing profit margin, the change is made only for units that are "active," i.e., have not yet been traded. The size of the adjustment is proportional to a learning rate parameter, similar to that used in Widrow-Hoff or in back-propagation learning. The difference between p_i and p_T is also stored for use at the next trade, when a further adjustment in the same direction is made, proportional to a separate "momentum" parameter. This is analogous to the use of momentum to speed up convergence in back-propagation learning.

If a sufficiently long time has passed without a trade taking place (1.0 seconds in our implementation), ZIP buyers and sellers adjust p_i in the direction of improving upon the best open competing bid or ask, if the i-th unit is still active. Finally, there is a global constraint that each p_i must always correspond to non-negative agent surplus, i.e. it must always be below the buyer's value, or above the seller's cost.

In all-agent tests, we find that homogeneous populations of ZIP traders achieve robust convergence to theoretical equilibria with high efficiency. Depending on the precise market rules and initial conditions, efficiencies ranging from 0.980 to 0.999 have been obtained with this strategy.

Gjerstad-Dickhaut (GD) Strategy

Gjerstad and Dickhaut [Gjerstad and Dickhaut, 1998] introduced a more sophisticated trading algorithm for buyers and sellers in the CDA, which we shall term "GD". They showed via simulation that a homogeneous population of such agents could attain high allocative efficiency and rapid convergence to the theoretical equilibrium price. A GD agent constructs an order and trade history H, consisting of all orders and trades occurring since the earliest order contributing to the Mth most recent trade. From the history H, a GD buyer or seller forms a subjective "belief" function $f(p)$ that represents its estimated probability for a bid or ask at price p to be accepted. For example, for a seller,

$$f(p) = \frac{AAG(p) + BG(p)}{AAG(p) + BG(p) + UAL(p)}, \qquad (1)$$

where $AAG(p)$ is the number of accepted asks in H with price $\geq p$, $BG(p)$ is the number of bids in $H \geq p$, and $UAL(p)$ is the number of unaccepted asks in H with price $\leq p$. Interpolation is used to provide values for $f(p)$ for prices at which no orders or trades are registered in H. The GD agent then chooses a price that maximizes its expected surplus, defined as the product of $f(p)$ and the gain from trade at that price (equal to $p - l$ for sellers and $l - p$ for buyers, where l is the seller cost or buyer value). Thus the algorithm does not require the knowledge or estimation of other agents' costs or valuations.

The original GD algorithm was developed for a market in which there was no queue, i.e. old bids or asks were erased as soon as there was a more favorable bid/ask or a trade. In our version of the CDA, unmatched orders can be retained in a queue, and therefore the notion of an unaccepted bid or ask becomes ill-defined. We addressed this problem by introducing into the GD algorithm a "grace period" τ_g. Unmatched orders were not included in H unless at least the grace period τ_g had passed since that order had been placed. Another modification to GD addressed the need to handle a vector of limit prices, since the original algorithm assumed a single tradeable unit.

We also found empirically that the original GD algorithm could behave pathologically, particularly for "fast" agents, which placed orders whenever an order or trade had been placed in the market. This often resulted in rapid bursts of orders and trades. If the last $2M$ orders resulted in M successful trades, then there were no unsuccessful orders in the history H. Laboring under the false assumption that *any* price would be accepted, the agents would then place absurdly low bids or high asks, gradually lowering them until trades finally began to occur once again, often in another burst. This cyclical behavior was associated with high trade price volatility.

To greatly reduce the chances of this occurring, we used a softer form of history truncation. All of the simple tally terms in Eq. 1 (and the analogous expression for buyers) were replaced by weighted sums that placed exponentially more emphasis on events that occurred most recently, and the truncation parameter M was increased to a much larger value. As hoped, soft truncation led to more sensible and stable pricing. It allowed the desired responsiveness to recent events, but also permitted information from old events to be used whenever there was insufficient information from recent events.

Homogeneous populations of modified GD agents also achieve robust convergence to equilibrium, with efficiencies comparable to those obtained by ZIP agents.

5 Experimental setup

Our experiments used the following CDA market rules:

1. The "NYSE" spread-improvement rule was in effect, requiring that new bids be priced higher than the current best bid (and the equivalent for asks). This conforms to prior CDA studies and is believed to facilitate convergence to equilibrium.
2. All orders were for a single unit only, and a player could have at most one open order. This was meant to simplify the task for both agents and humans, and again conforms to prior CDA studies.
3. Submitted orders remained open until they were traded or the period ended.
4. Submitted orders could be modified (subject to the NYSE rule), but not withdrawn.
5. Trades occurred when the best bid and best ask matched or crossed in price. If they crossed (i.e., bid price exceeded the ask price), the trade price was the price of the order submitted first.
6. At the start of each period, players were given a fresh supply of cash or commodity.

Each player was given a list of 8-14 limit prices for the units to be traded, arranged in order from most to least valuable (i.e., the buyer values decreased and the seller costs increased). Roughly half of the players' units were tradeable for positive surplus at equilibrium. The limit prices were generated from a base set of three linear schedules in which each successive unit increased in cost or decreased in value at a constant rate. These rates varied in the three schedules; however, the total theoretical surplus was designed to be about the same in each. Each human had an agent counterpart with the same role and the same limit prices, and hence the same theoretical surplus. The total theoretical surplus was designed to split about evenly between buyers and sellers.

An experiment consisted of 15-16 trading periods of 3 minutes each. Every 4-5 periods, each player's limit prices were changed by rotating the three limit price schedules (e.g., seller A received seller B's previous schedule, B received C's, and C received A's) and adding or subtracting a constant value to all limit prices, so as to change the equilibrium price.

Our target configuration for the experiments consisted of 6 agents and 6 humans[2], both split evenly between buyers and sellers. In each experiment, all agents used the same bidding strategy—either ZIP or GD—and were all either the "Fast" or "Slow" variant. Human subjects in four experiments were undergraduates from local colleges; in two others, they were employees of IBM Research. Before the start of each experiment, subjects received instruction on the auction rules and the profit objective, and practiced using the GUI. No discussion of bidding strategies was given. Subjects were told they would receive cash payments proportionate to profits earned in the auction; the conversion factor was set so that the expected payouts were $\sim \$50 - 60$ per player.

[2]In some experiments, one or more of the scheduled subjects failed to appear, resulting in an asymmetric market with more agents than humans.

6 Experimental results

Table 1 summarizes the results of the six agent-human CDA experiments. Several noteworthy findings were obtained. First, there were significant interactions and trades between agents and humans, even though the agents were potentially much faster. Roughly 30% or more of all trades were between agents and humans. This is a reasonable fraction of the naive expectation of 50% if any trade partner (agent or human) is equally likely, and shows that the laboratory markets did genuinely test agent-human interactivity, as opposed to merely creating two non-interacting sub-markets operating on different time scales.

Second, when considered as a group, the agents outperformed the humans in all six experiments: the total surplus obtained by agents was on average $\sim 20\%$ more than the total human surplus. This was true for both fast and slow agent populations, showing that speed was not the sole factor accounting for the agents' edge in performance. In terms of average efficiency, agents in aggregate tended to achieve greater than 100% efficiency, which necessarily implies that they were exploiting human errors or weaknesses. Humans, on the other hand, tended to score in the range of $\sim 0.92 - 0.96$, and on two occasions they did much worse. To check our market design, we ran a baseline experiment in which all 12 traders were human, measuring an efficiency of 0.96, which is consistent with what is typically found in all-human CDA experiments. The fact that humans play better against other humans than they do against agents corroborates the evidence that, as a group, the agents are stronger players.

Third, as in prior all-human CDAs, human performance tended to improve during the course of an experiment, as the subjects became more familiar with the GUI and the market behavior, and got a better idea of how to execute a good bidding strategy. Nevertheless, we still found a consistent edge in agent surplus over human surplus by about $5 - 7\%$ in the final periods of each experiment.

Finally, although it is not documented in Table 1, our agent-human markets tended to have a lopsided character in which either buyers consistently exploited sellers, or vice versa. The previously-described scalloping behavior was observed to be more pronounced and longer-lasting than in prior all-human or all-agent CDAs. This will be discussed in detail in Sections 6.1 and 6.2, which examine two specific experiments with different agent bidding strategies (Fast GD and Slow ZIP), yielding distinctly different market dynamics.

6.1 Fast GD Agents vs. Humans

Figure 2 shows the trading activity in experiment Oct25, which was conducted over 16 periods divided equally among four phases with shifts in limit prices. In each period, the time series of trades tends to show scalloping, as trade prices converge towards the equilibrium price p^*. Such scalloping was not observed in agent-only CDA experiments with GD Fast (or Slow) agents. In this experiment, the buyers were able to extract more surplus from the market than the sellers as most trades occurred below p^*, a fact that is also reflected in the average efficiency measures. However, the differences in surpluses (and efficiencies) between the two sides of the market shrank over time, due to improving overall market efficiency,

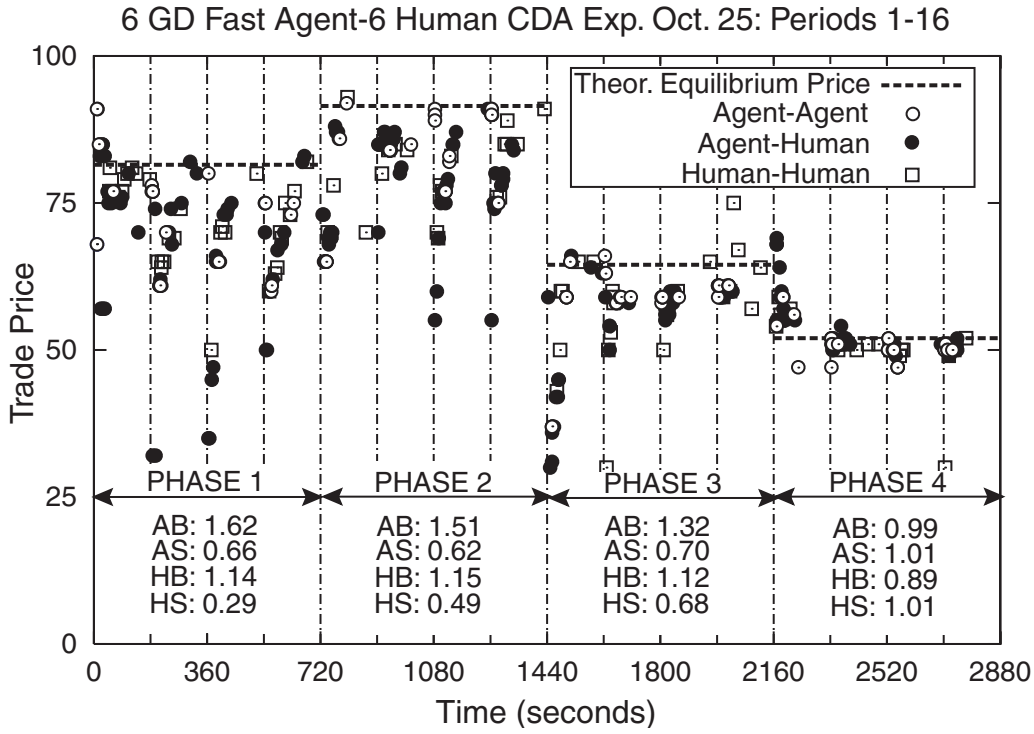


Figure 2: Trade price vs. time for experiment Oct25 with 6 GD fast agents and 6 humans. The vertical dashed lines indicate the start of a new trading period. The 16 trading periods were divided into four phases, each with its own set of limit prices. In each phase, p^* is shown by the horizontal dashed lines. Trades between two agents are shown with open circles, between two humans with open squares, and between an agent and a human with solid circles. The labels AB, AS, HB, and HS refer to the average efficiency of Agent Buyers, Agent Sellers, Human Buyers, and Human Sellers, respectively.

which we attribute to human improvement during the course of the experiment. Note that in all four phases, the agent buyers and the agent sellers are able to extract more surplus than their human counterparts.

Figure 2 also shows that most of the lowest-priced trades, occurring well below p^*, were between agents and humans. An inspection of experimental records reveals that these trades were mostly between human sellers and agent buyers. Apparently the human sellers were consistently offering excessively low asks, and the agent buyers were able to pounce on such mistakes more quickly than their human counterparts.

6.2 Slow ZIP Agents vs. Humans

Figure 3 shows the trading activity in the third phase of experiment Oct24a with 6 ZIP slow agents and 6 humans. The figure shows several interesting features. First, pronounced and repeated scalloping in trade prices is evident, with buyers extracting much more surplus than sellers. This is surprising since such large scalloping is not seen in markets with only ZIP agents. Second, in each period, trades typically tended to occur first between agents, then between agents and humans, and finally between humans. Third, although the agents as a group outperformed the humans, agent sellers actually obtained less surplus than human sellers.

Subsequent interviews with human subjects helped to explain the behavior and ultimately reveal a weakness in the ZIP strategy. It turned out that two of the human sellers in this experiment consistently followed a 'fixed-profit-ratio' strategy, that is, their asking price for each unit was a fixed percentage greater than its cost. These sellers repeatedly submitted asks at prices much lower than p^* and most often, agent buyers quickly accepted such offers. Having traded their units at extremely low prices, the ZIP buyers ignored subsequent trades at higher prices, and maintained very low bid prices at the start of the next period. The resulting bid-ask spread at the start of each period was centered well below p^*, and once the human sellers made a few low-priced sales, the ZIP sellers, which were waking up on trades, began quickly dumping their inventory at very low prices. Hence we attribute the unusual market behavior, and the performance ranking of the agent and human traders, to a set of odd interactions between the ZIP strategy and a specific non-optimal human strategy.

To test our hypothesis, we performed separate all-agent experiments with a fixed-profit-ratio agent in a population of only ZIP agents. The resulting dynamics exhibited large scallops in trade prices similar to those in Figure 3. We also

Experiment				Agent			Human		
ID	# Periods	# Trades	Interaction	Strategy	Surplus	Efficiency	# Traders	Surplus	Efficiency
Oct17	15	412	0.38	GD Fast	11058	1.016	5	6991	0.927
Oct18	15	504	0.29	ZIP Fast	11069	1.028	6	7023	0.652
Oct23	16	320	0.33	GD Fast	10495	0.999	3	4582	0.965
Oct24a	16	455	0.48	ZIP Slow	10696	1.032	6	9490	0.916
Oct24b	9	261	0.42	GD Fast	6808	1.026	6	6353	0.958
Oct25	16	433	0.49	GD Fast	12159	1.052	6	9708	0.840

Table 1: Summary of the six agent-human CDA experiments. For each experiment, the table presents: the number of trading periods, the total number of successful trades, the fraction of trades between agents and humans, the bidding strategy employed by all six agents, the number of human traders, and the aggregate agent and human performances in terms of total surplus accumulated over the entire experiment and the average efficiency.

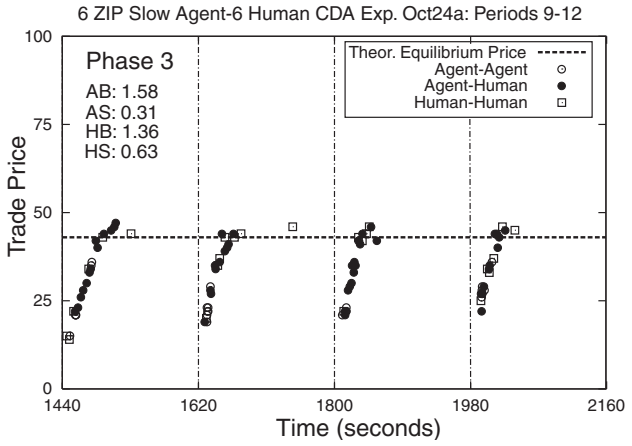


Figure 3: Trade price vs time for experiment Oct24a with 6 ZIP slow agents and 6 humans. Trading activity in periods 9-12 (out of 16 total) is shown. Other details are similar to those in Figure 2.

implemented a modification to the ZIP strategy that is more reluctant to lower its profit margin based on where trades occurred in the last period. Preliminary results show that the modified ZIP agents retain high efficiency and are not easily misled by the fixed-profit-ratio agent.

7 Conclusion

Given the simplicity of our agent bidding strategies, we are encouraged by the results of these first-ever tests against human subjects. (We remark that, while substantial agent-human interactions have already occurred in financial markets, such interactions have an unknown and uncontrolled nature.) Our agents make relatively simple time-independent price calculations, based on established algorithms, and their timing decisions are equally simple, yet they are able to outperform non-expert human subjects by a clear margin. It would be interesting to test our bidding agents against a higher caliber of human opposition, e.g., professional equities or commodities traders. We suspect that such opponents would uncover weaknesses in the agent strategies, and that this would eventually lead to significant algorithmic improvements in the strategies. We are optimistic that CDA strategies can be improved to the point where they outperform all human opposition by making better price inferences based on market history, and by taking time remaining into account in pricing and timing decisions.

Several aspects of the market behavior in our experiments differed from prior studies of all-agent or all-human traders. Convergence to equilibrium in our experiments was generally slower than in prior studies, and in two experiments, there was no evidence of convergence by the end of the experiment. We observed scalloped price trajectories that were more pronounced and longer lasting than seen previously. Also, our markets tended to be much more lopsided (either buyers systematically exploiting sellers, or vice versa) than in earlier studies. Such novel market phenomena merit further investigation: they might be due to specifics of our market design, or they may be more general outcomes of agent-human interactions. Hence it will be important to conduct further agent-human experiments in other types of markets, possibly including greater complexity and real-world detail. Candidates include combinatorial auctions, TAC-type markets [Wellman *et al.*, 2001], and more realistic models of financial markets. The development and deployment of effective automated trading strategies in such markets would have immense practical importance, and could mark the beginning of a large-scale introduction of economic software agents into the world economy [Kephart *et al.*, 2000].

While our results are preliminary, some aspects of our findings may be indicative of what one can expect to occur more generally as economic software agents are developed for real-world markets. We suspect that, in many real marketplaces, agents of sufficient quality will be developed such that most agents beat most humans. A significant component of their advantage will come from their ability to initiate actions, and to react to market events, much faster than humans. As a result, there will be significant economic incentive for humans to employ agents to act on their behalf. Then the competition between agents and humans will evolve into a competition among agents.

Acknowledgments

The authors are especially indebted to Steven Gjerstad, who made several invaluable contributions to these experiments.

Steven co-authored the paper that first described the GD bidding algorithm, which we adapted and extended to handle different market rules. Steven was also responsible for the design of the Watson Experimental Economics Laboratory (WEEL) where the experiments were conducted; for the design of the double auction market rules; for the design of the GEM system that was used as the electronic auctioneer and the auction clients for human traders; and for recruiting most of the experimental subjects.

We would like to thank several other people for their significant contributions. Jason Shachat helped develop the experimental protocol and supervise some of the experiments. Amit Shah helped implement the messaging link between the GEM and Magenta systems. Oconel Johnson, system administrator for WEEL, helped ensure that the GEM hardware and software were ready for the experiments. Weng-Keen Wong and Jonathan Bredin developed the Magenta message handling components for the agents and a Magenta auctioneer used for stand-alone testing, and they implemented and tested several bidding algorithms. David Levine resolved several bugs and inefficiences in the GEM-Magenta link, and was an invaluable resource on architectural questions.

We would also like to thank the many students and IBM researchers who participated in the experiments, including some early trial experiments not reported here. Finally, we would like to acknowledge the support of the IBM Institute for Advanced Commerce.

References

[Cliff and Bruten, 1997] D. Cliff and J. Bruten. Minimal-intelligence agents for bargaining behaviors in market-based environments. Technical Report HPL-97-91, Hewlett Packard Labs, 1997.

[Gjerstad and Dickhaut, 1998] S. Gjerstad and J. Dickhaut. Price formation in double auctions. *Games and Economic Behavior*, 22:1–29, 1998.

[Kephart *et al.*, 2000] J. O. Kephart, J. E. Hanson, and A. R. Greenwald. Dynamic pricing by software agents. *Computer Networks*, 32:731–752, 2000.

[Preist and van Tol, 1998] C. Preist and M. van Tol. Adaptive agents in a persistent shout double auction. In *Proceedings of the First International Conference on Information and Computation Economies*, pages 11–18. ACM Press, 1998.

[Rust *et al.*, 1992] J. Rust, J. Miller, and R. Palmer. Behavior of trading automata in a computerized double auction market. In D. Friedman and J. Rust, editors, *The Double Auction Market: Institutions, Theories, and Evidence*, Redwood City, CA, 1992. Addison-Wesley.

[Schaeffer, 2000] J. Schaeffer. The games computers (and people) play. In M. Zelkowitz, editor, *Advances in Computers 50*, pages 189–266. Academic Press, 2000.

[Smith, 1962] V. L. Smith. An experimental study of competitive market behavior. *Journal of Political Economy*, 70:111–137, 1962.

[Smith, 1982] V. L. Smith. Microeconomic systems as an experimental science. *American Economic Review*, 72:923–955, 1982.

[Wellman *et al.*, 2001] M. Wellman, P. Wurman, K. O'Malley, R. Bangera, S. Lin, D. Reeves, and W. Walsh. Designing the market game for a trading agent competition. *IEEE Internet Computing*, pages 43–51, March/April 2001.

MULTI-AGENT SYSTEMS

USER INTERFACES

Usability Guidelines for Interactive Search in Direct Manipulation Systems

Robert St. Amant and Christopher G. Healey
Department of Computer Science
North Carolina State University
EGRC-CSC Box 7534
Raleigh, NC 27695-7534, U.S.

Abstract

As AI systems make their way into the mainstream of interactive applications, usability becomes an increasingly important factor in their success. A wide range of user interface design guidelines have been developed for the direct manipulation and graphical user interface conventions of modern software. Unfortunately, it is not always clear how these should be applied to AI systems. This paper discusses a visualization assistant, an e-commerce simulation domain we have applied it to, and the guidelines we found relevant in the construction of its user interface. The goal of this paper is to explain how an interactive system can incorporates search-based intelligent behavior while still respecting well-established rules for effective user interaction.

Keywords: user interfaces, expert systems, simulation

1 Introduction

Designing a good user interface may appear to be straightforward, especially with the help of a user interface builder, but this ease is deceptive. The construction process involves much more than adopting the surface conventions of graphical user interfaces, mapping inputs to menus and type-in boxes, outputs to icons and related graphics. Good graphical user interfaces also follow heuristic guidelines that govern almost every aspect of design, from the manipulation properties of widgets to the decomposition and organization of tasks to the overall visual layout of an interface [Dix *et al.*, 1998]. Determining how a general guideline should be applied in any given situation is notoriously difficult.[1]

This is a newly important issue for AI, as search-based techniques find their way into conventional interactive applications. Interactive AI systems usually depend on *direct manipulation* in some form, by virtue of their integration into modern software environments. Direct manipulation systems rely on continuous representation of objects of interest, physical actions or labeled button presses for commands, and rapid, incremental, reversible operations with visible feedback [Shneiderman, 1998].

Unfortunately, intelligent behavior sometimes makes demands on the interaction process that do not naturally fit into a direct manipulation framework [Shneiderman and Maes, 1997]. AI systems have traditionally treated interaction with a user as a kind of *communication*. In contrast, direct manipulation relies on what Hutchins calls *model world* interaction [Hutchins, 1989]: the interface provides an environment and a set of tools that the user directly applies to reach a problem solution. More concisely, the user acts through a direct manipulation interface, rather than talks to it. The difference between the communication and model world paradigms is more than skin deep, in that what constitutes an effective set of rules for interaction under one paradigm may be drastically inappropriate under the other. For example, in a collaborative problem-solving process, it is common for agents to negotiate about the appropriate means to solve a problem. In contrast, tools do not negotiate with their users; they simply perform their assigned tasks. If the effectiveness of an interactive system depends on simple tools that give the user predictable, consistent behavior, then redesigning the tools so that they can communicate and negotiate with the user about their use is likely to degrade the overall usability of the system.

This is not to say that agents cannot behave as tools, or that tools cannot be extended to incorporate some agent-like behavior (e.g. [Anderson *et al.*, 2000; Horvitz, 1999; Lieberman, 1995; Maes, 1994]). On the contrary, we believe that the line between agents and tools will continue to blur. A key issue in this evolution is whether we understand the differences in user interaction that the agent-based and tool-based approaches require. With this understanding we can make better decisions about how conventional user interface guidelines should be adapted to unconventional (from a human-computer interaction perspective) application properties, such as a reliance on search.

This paper lays out a set of design guidelines for incorporating a search process into a conventional direct manipulation interface. We describe several design principles from the HCI literature and work out their general implications for systems that provide intelligent assistance through search. We further discuss how these implications affect the design of a search-based system for visualization.

[1] Even commercial applications sometimes suffer from serious usability problems; examples are found in textbooks (e.g. [Dix *et al.*, 1998]), popular works (e.g. [Raskin, 2000]), and online (e.g. `http://www.iarchitect.com/mshame.htm`), as well as in the technical literature (e.g. [Thimbleby, 2000]).

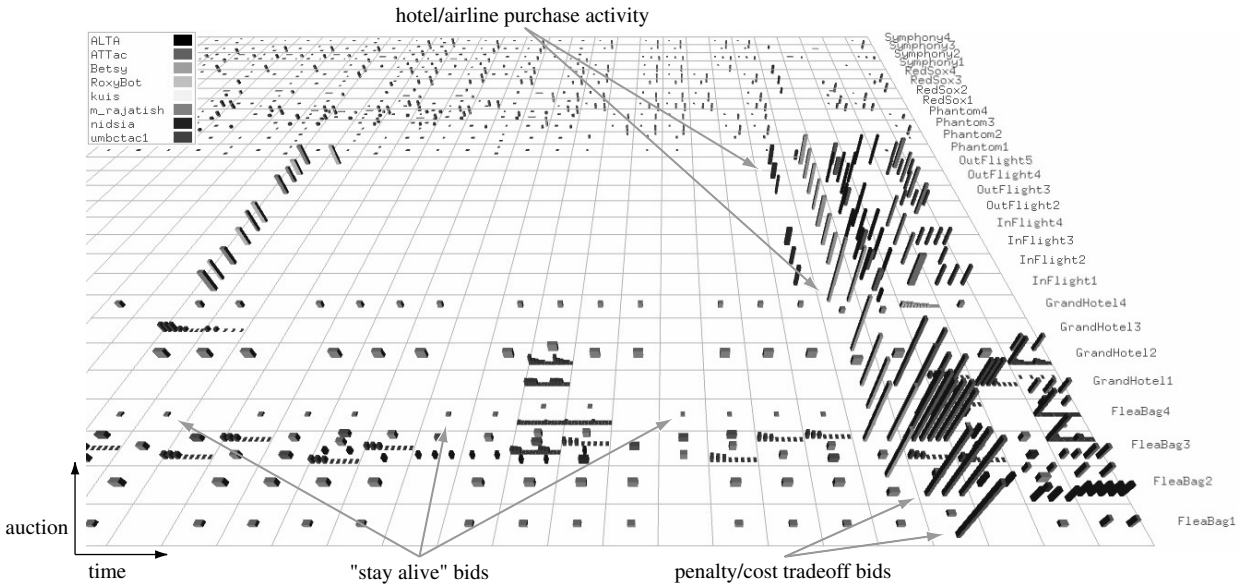


Figure 1: ICMAS '00 TAC data visualization (in grayscale), with *agent ID* $\rightarrow$ *color*, *price* $\rightarrow$ *height*, *quantity* $\rightarrow$ *width*

2 A visualization domain

We begin with a example, using a problem domain that illustrates our motivation as well as producing results of general interest in AI. ViA is an intelligent assistant for building scientific visualizations. Scientific visualization is the conversion of collections of strings and numbers, or datasets, into images that allow viewers to perform visual exploration and analysis. Very large datasets with millions of data elements representing multiple independent data attributes are not uncommon. The challenge is to design methods that present some or all of this information simultaneously in a single display, without overwhelming a viewer's ability to make sense of the resulting images. The choice of which visual features to use (e.g. color, size, or contrast) to represent each data attribute is called a data-feature mapping. ViA takes as input a dataset, a short description of the dataset's properties, and a set of viewer-defined analysis tasks, and produces as output a set of appropriate data-feature mappings.

One current testbed application, the Trading Agent Competition,[2] illustrates the goal of ViA. The TAC is a simulated e-commerce auction environment run on the Michigan Internet AuctionBot platform.The AuctionBot is a TCP-based auction server that implements numerous types of auction rules. This allows the simulation of a wide variety of market games. Intelligent auction agents are designed and tested within these markets to study different buying and selling strategies.

During the TAC, each agent acts as a "travel advisor" whose goal is to assemble a travel package for eight fictitious customers. A travel package consists of a round-trip flight from TACtown to Boston, a hotel reservation, and tickets to entertainment events (a baseball game, the symphony, and the theater). Customers specify preferences for the different aspects of their trip: which days they want to be in Boston, the type of hotel they prefer (economy or luxury), and the entertainment events they want to attend. Obvious dependencies must be met; for example, customers need hotel rooms for the duration of their trip, and can only attend entertainment events during that interval. The goal of the agent is to maximize the total satisfaction of its customers (i.e., the sum of their utility functions).

ViA was used to identify effective real-time visualizations for agent activity in a TAC run during the ICMAS '00 conference. Five separate attributes were selected by hand for visualization: the *time, auction ID, agent ID, price*, and *quantity* for every bid made during the simulation. *time* and *auction ID* were used to define a bid's x and y-position on a two-dimensional grid. Perceptual texture elements (or pexels [Healey and Enns, 1999]) that vary in their color (combined hue and luminance), height, density, and regularity of placement were used to represent the remaining attributes: *agent ID*, *price*, and *quantity*.

TAC competitors, acting as domain experts, assigned normalized importance weights of 1 (very important) to *agent ID*, and 0.5 (somewhat important) to *quantity* and *price*. They also defined the analysis tasks (searching for specific agents, identifying price boundaries, and estimating the relative frequency of particular quantities) they wanted to perform. ViA automatically identified additional properties like the spatial frequency and value range of each attribute in the dataset. This application-independent information is used together with a collection of perceptual guidelines to select the mappings that are most appropriate for TAC visualizations.

Based on dataset properties and viewer-imposed restrictions, ViA produced several perceptually salient visualizations. Figure 1 shows a modified version of a ViA-generated mapping, with *agent ID* $\rightarrow$ *color*, *price* $\rightarrow$ *height*, and *quan-*

[2] `http://tac.eecs.umich.edu`

tity → *width*. Rectangular towers are used to represent each bid made during the game. *time* and *auction ID* define a tower's location on the underlying grid. Time increases from left to right along the horizontal axis. Each row corresponds to a separate auction. A tower's color, height, and width show the bid's *agent ID*, *price*, and *quantity*, respectively. Different colors identify bids by different agents. Buy bids lie above the grid, while sell bids lie below the grid; a higher buy (or sell) price increases the height of the tower. Bids for larger quantities produce wider towers. Simultaneous bids made by an agent in a particular auction are displayed as a horizontal row of towers in the appropriate grid cell (each with its height and width defined by the particular bid's *price* and *quantity*). Bids made by different agents at the same time in the same auction are shown as rows of towers drawn on above another in a common cell. This arrangement uses spatial density to represent the level of bid activity at different locations in the game.

Many aspects of the agents' strategies and game play can quickly be identified using such visualizations. For example:

1. Some agents periodically made very low buy bids for hotel rooms to ensure the hotel auctions "stay alive" (hotel auctions automatically close after a period of inactivity).
2. Most agents deferred purchasing hotel rooms and airline tickets until just before the simulation ended, judging there was no advantage to early purchase (particularly for hotel rooms, where attempts at early purchase can drive up the final price).
3. If hotel rooms for a given customer cannot be found, the customer's entire trip is canceled, and the agent is penalized the cost of any airline and entertainment tickets they may have purchased on the customer's behalf. Some agents estimated the marginal cost c of this penalty, then made late bids for hotel rooms at a buy price of c. These agents decided that paying c for a hotel room was no worse than paying a penalty of c for unused airline and entertainment tickets. More importantly, there is a good chance that the hotel rooms will sell for less than c. If this happens, the agent will make a profit relative to the scenario of not securing the hotel room.

ViA casts scientific visualization as a straightforward search problem: finding a mapping between dataset attributes and visualization features that respects dataset and user constraints. ViA's exploration, at its core, is an incremental best-first search. The evaluation function is composed from the results of a set of critics. A critic exists for each visual feature available for use in a mapping (e.g. a hue critic, a height critic, and so on). The critic examines the data attribute associated with its visual feature, and reports on its perceptual correctness [Healey and Enns, 1999]. Critics also generate search operators for improving particular associations. ViA uses the critics' results to compute an overall evaluation of its current mapping, and to extend the search in the direction of modifications that should produce better mappings.

Our discussion up to this point has skirted the issue of user interaction. Even for the TAC domain, which poses a very small problem in computational terms, the design of an interface for ViA must address a number of nontrivial questions. How can user preferences be taken into account to guide the search process? What kind of incremental feedback should the system provide, for lengthy processing times? Should the user see all of the solutions ViA generates, or only the best? Answers are provided in the HCI literature, but require interpretation to be applied correctly.

3 Interaction design for an AI system

The HCI literature contains dozens of general principles for user interface design, leading to thousands of detailed guidelines. For example, Smith and Mosier produced an early set of over 900 guidelines for text-based interfaces [Smith and Mosier, 1986]. The sophisticated user interfaces of Macintosh and Windows systems are driven by a small set of high level principles, including metaphor, direct manipulation, user control, and consistency, that expand to much more detailed guidelines. We draw on a recent concise summary of concepts due to Dix et al [1998]. The category of *learnability* in an interface includes predictability, synthesizability, familiarity, generalizability, and consistency. *Flexibility* includes dialog initiative, multi-threading, task migratability (i.e., from the user to the system, and vice versa), substitutivity of input and output forms, and customizability. *Robustness* includes observability, recoverability, responsiveness, and task conformance. Some of these concepts are applicable to all user interfaces. Those we consider in this section pose some novel requirements that we have not seen explicitly compiled and discussed in the AI or HCI literature.

Let's consider a hypothetical interactive AI system, a generalization of the ViA system, to see how these guidelines should be interpreted. The system is given a problem that it must solve through search. Its evaluation function, while accurate to an approximation, is incomplete; the search process requires input from the user to reach the best results (e.g., prefer *this* data-feature mapping to *that*, due to domain-specific interpretations of color.) This means that the system must show the user some representation of the state space, in order to elicit feedback and guidance. This interaction requirement in turn means that a significant part of the system's design must be devoted to managing interaction with the user. Unlike conventional applications, an AI system takes actions that are determined as much by the properties of the state space as by the actions of the user. The interaction is thus more likely to be opportunistic, less likely to be predictable with respect to the type of information exchanged, the duration of the entire task, how control of initiative shifts between the system and the user, and related properties.

Given this broad description, we now turn to an interpretation of the usability concepts listed above, organized into four broad areas. Following these guidelines generally improves conventional interfaces; we show how they can be applied to interactive AI systems.

Incremental processing is an important property for an interactive AI system, if for no other reason than that the user must be given some opportunity to contribute to the problem solving process. This general point has several related components.

Responsiveness. An interface is responsive when the response times of its operators match the user's expectation. Some (though not all) usability studies have found that consistency in response time is preferable to raw speed [Myers, 1985]. For example, users will generally prefer a constant response time (i.e. search duration) of, say, five seconds, to responses with a mean time of four seconds but varying between zero and eight seconds. For a search-based system, this suggests a design that combines anytime processing with continuous computing concerns [Horvitz, 1997].

Task migratability. Task migratability is supported when problem solving responsibility can be handed off from the user to the system, and vice versa. This suggests that the system and the user should have access to a common set of search operators (though user operators will often be abstractions or compositions of system operators.) This kind of facility is supported, for example, by scripting and end-user programming in conventional software, and by some systems for programming by example (PBE) [Lieberman, 2001]. Migratability is impaired if a search-based system can reach states that the user cannot reasonably evaluate, or if the user would like to override specific system behavior but cannot.

Observability implies that the user is able to infer internal properties of the system's state from its external representation. An important aspect of observability in an AI system is making clear the "maturity" of a solution. The pitfall to avoid is the system prematurely focusing the user's attention on an early solution that is likely to be superseded by a later incompatible solution. This can be viewed from the system side as a search horizon issue, from the user side as a potential anchoring problem.

Dialog initiative. An interface respects dialog initiative when problem-solving initiative can shift between the system and the user to follow the task. As with task migratability, mixed dialog initiative should be supported by presentation of information at an appropriate level of detail for the user to make a meaningful contribution.

Adaptation refers to the dynamic adjustment of the system's behavior to the user's actions. This adaptation can occur at the direction of the user or automatically.

Customizability entails that an interface be adaptable to the abilities and needs of the user. This kind of customization, for a search-based system, can be thought of as modification of preferences that influence the system's operational characteristics. Candidates for customization include the number of states incrementally searched, and the number of potential solution states retained and presented.

Search adaptation is a more important point. Suppose that the system presents the user with the current best state s_i, as determined by its evaluation function f. The user reviews this information and decides that a modification is appropriate, leading to a state s_{i+1} that the system has already considered internally and given a lower value. When the system resumes its search, it must modify f or some state property to avoid again indicating that s_i is a better solution. A further useful capability is generalization of the differences between s_i and s_{i+1} to support comparisons of other states that have not yet been evaluated or presented; some PBE systems have this capability.

Coherence. Although direct manipulation systems tend to support a style of interaction in which operator sequences are short and goals interact as little as possible, an incremental search process necessarily involves maintenance of context between exchanges with the user.

Predictability and consistency are two important aspects of coherence. An interface is predictable when a user can determine, from a specific operation in a given state, what the consequences will be. Consistency promotes predictability. Direct manipulation systems tend to present a comprehensive visual environment to the user, with the assumption that the user can judge relevance better than the system. In an incremental search, a system might reserve some information that is simultaneously less likely to change and less likely to be immediately relevant. Instead of giving the user a complete representation of the best n states, for example, the system might limit its display to a few selected properties, leaving itself "wiggle room," space to maneuver.

A related issue deals with predictability in response to user actions. To continue the search adaptation example, suppose that the user has selected a specific operator p to improve state s_i, and that p has been effective in the past, but this time it is not. If the reason for this is not apparent (an observability issue), the user's confidence in the effectiveness of p or the system's evaluation function may suffer. Additional explanation may be warranted to improve predictability.

Support for user orientation. One of the difficulties implied in the foregoing discussion is that of managing interaction between the system and the user, such that each party can make real contributions. Many of these difficulties can be understood in terms of user orientation during a navigation process [Kim and Hirtle, 1995].

Reachability between states in an interface generalizes the idea of recoverability; it implies that from any state, any other state can be reached, in particular non-error states from error states. For an intelligent interactive system this goes beyond the completeness of a search algorithm. In addition it means that the user can recall and re-visit past states, to review and re-evaluate earlier decisions. Navigational support (e.g. generation of landmarks) can improve orientation for this task.

Synthesizability can be understood as the ease with which the user can form a conceptual model of the problem-solving process. In an interactive AI system, part of synthesizability means that the system partitions the state space so that the user can grasp individual portions more easily. In navigation terms, this kind of synthesizability is supported by region differentiation.

Property	*ViA mechanism*
Responsiveness	Response time tailored to platform
Dialog initiative	User controlled incremental search
Task migratability	External availability of operators, executable between search epochs
Customizability	User-editable preferences for search step size and filtering of results

Table 1: Usability properties in ViA search

Property	*ViA mechanism*
Predictability	Clustering and filtering of best states; presentation of partial state information
Synthesizability	Allowing user to explore "nearby" and "distant" solutions of comparable value, with region partitioning by windows
Observability	Progress messages and navigation support
Reachability	Navigation support

Table 2: Usability properties in ViA presentation

4 Interaction with ViA

In the user interface to ViA we are exploring the implications of these guidelines. A functional prototype is shown in Figure 2, with an intermediate step in the search for data-feature mappings visible. (This section describes a weather dataset rather than the TAC dataset, which is too small to exercise some of the capabilities of the interactive system.) ViA's interface design accounts for most of the concerns identified in Section 3, although not with complete generality. Relevant design features can be divided into those that affect the search directly, as summarized in Table 1, and those that affect presentation and user interaction, as shown in Table 2. These design features will appear to be straightforward, even obvious, for the most part, but it is surprising how often they are neglected in interactive AI systems; hence the need for guidelines.

ViA's search is incremental, with a default but editable response time of about a second. The search step size is computed automatically when the user loads an initial dataset description, by running the critics on the dataset to gain timing estimates for the current platform. The user can incrementally modify the mappings that the system returns; these editing capabilities provide most of the functionality of ViA's search operators.

In ViA's domain, a search usually produces several candidate solutions with essentially equal evaluation results. The system maintains a list of the best ($n = 20$) solutions it has encountered in its search, and performs a simple clustering on the contents of the list at the end of each search epoch. If a cluster is found among the top-ranked solutions in which the majority of attributes are assigned to the same features, then the remaining, differing assignments are stripped out and only the common assignments of the solutions are presented. This capability is intended to focus the user's attention on the more influential assignments; whether the filtering occurs at all, however, is under user control.

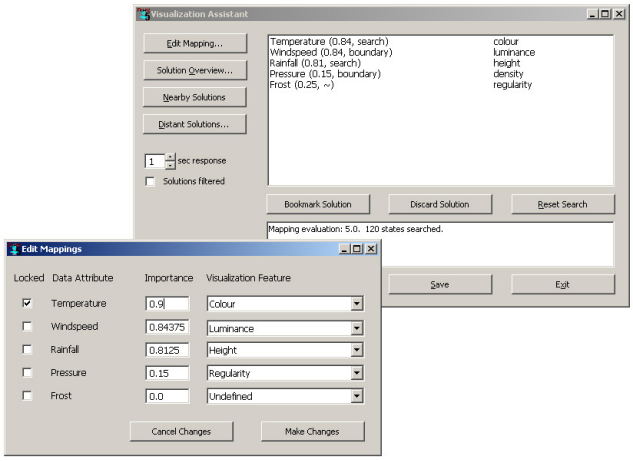


Figure 2: ViA user interface

ViA's search can be directed toward solutions that are near the currently displayed mapping, or those that are distant. When the user chooses to search nearby solutions, the system selects the best solution it finds that is nearest to the solution most recently presented. Distant solutions are handled by selecting farthest solution among the top candidates. This organization of search results is a compromise between showing all promising solutions at once and showing them one at a time in an arbitrary order. The nearby/distant distinction allows the user to conceptually organize the space of mappings into distinct regions during its traversal. Toward this end, a distant solution operation generates results in a new window, to create a new visual "context" for the search. As another navigational aid, the user can bookmark or discard individual solutions to impose further structure on the space.

A global search overview shows all of the presented solutions, across all nearby and distant searches. Currently the overview information is presented as text in outline form, but the intention is to build an annotated graphical representation of the search space, to show the non-linear paths that the user may have taken through it. Other areas of incomplete or missing functionality include the adequacy of explanations provided to the user (explanations currently only describe critic results and the number of states searched), the completeness of the set of search operators available to the user (discretization and task removal operators are missing), and functionality related to search adaptation.

The interface that has resulted from our work supports a more flexible, less burdensome interaction with the search component of ViA than is provided by conventional means (e.g., interaction through a command line, within a text editor, or through a simpler graphical interface dialog.) As with many systems in the intelligent user interfaces and HCI literature, however, the interactive aspects of ViA outlined above have not been evaluated in a formal sense. The application discussed here should not be treated as validation of ViA's interface design. Rather, our examples act as illustrations and

an early means of formative evaluation. Evaluating a user interface of any complexity is an extensive undertaking and remains yet to be done for ViA.

5 Discussion

Our work is partly inspired by earlier work on interface softbots, which operate through the user interface of an application, rather than a programmatic interface [St. Amant and Zettlemoyer, 2000]. Interface softbots are motivated by a claim that characteristic properties and behaviors of a graphical user interface can be exploited by planning agents, due to a similarity between planning assumptions and user interface guidelines [St. Amant, 1999]. In this paper we take a more conventional route in showing how such interface guidelines can be applied to improve interactive AI systems.

Otherwise, usability for interactive AI systems has been a small but active area of research [Hendler and Lewis, 1988; Höök, 2000; Kaasinen, 1998]. Horvitz provides a representative study, presenting a set of principles for mixed-initiative user interfaces [Horvitz, 1999]. As is commonly the case, these extend beyond conventional guidelines for direct manipulation systems, encompassing such issues as social behavior and the explicit use of dialog. Our intention is not to describe new guidelines for new technology, but rather to clarify how new techniques can be fit into an existing interaction paradigm, and to explain how existing guidelines apply.

Acknowledgments

Thanks to Peter Wurman for his contribution of TAC material and to three anonymous reviewers for helpful comments on the draft of this paper. This research was partially supported by the National Science Foundation, award numbers IIS-9988507 and IIS-0083281. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright notation hereon.

References

[Anderson *et al.*, 2000] David Anderson, Emily Anderson, Neal Lesh, Joe Marks, Brian Mirtich, David Ratajczak, and Kathy Ryall. Human-guided simple search. In *Proceedings of AAAI*, pages 209–216. AAAI Press, 2000.

[Dix *et al.*, 1998] Alan J. Dix, Janet E. Finlay, Gregory D. Abowd, and Russell Beale. *Human-Computer Interaction*. Prentice Hall, 2nd edition, 1998.

[Healey and Enns, 1999] Christopher G. Healey and James T. Enns. Large datasets at a glance: Combining textures and colors in scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(2), 1999.

[Hendler and Lewis, 1988] James Hendler and Clayton Lewis. Introduction: Designing interfaces for expert systems. In James Hendler, editor, *Expert Systems: The User Interface*, pages 1–14. Ablex, 1988.

[Höök, 2000] Kristina Höök. Steps to take before intelligent user interfaces become real. *Interacting with Computers*, 12(4):409–426, 2000.

[Horvitz, 1997] Eric Horvitz. Models of continual computation. In *Proceedings of AAAI*, 1997.

[Horvitz, 1999] Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of CHI'99*, pages 159–166, 1999.

[Hutchins, 1989] Edwin Hutchins. Metaphors for interface design. In M. M. Taylor, F. Neel, and D. G. Bouwhuis, editors, *The Structure of Multimodal Dialogue*, pages 11–28. North-Holland, Elsevier Science Publishers, Amsterdam, 1989.

[Kaasinen, 1998] Eija Kaasinen. Usability issues in agent applications: What should the designer be aware of. Technical report, USINACTS, 1998.

[Kim and Hirtle, 1995] Hanhwe Kim and Stephen C. Hirtle. Spatial metaphors and disorientation in hypertext browsing. *Behaviour and Information Technology*, 14(4):239–250, 1995.

[Lieberman, 1995] Henry Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 924–929, 1995.

[Lieberman, 2001] Henry Lieberman, editor. *Your Wish Is My Command: Programming by Example*. Morgan Kaufmann, San Francisco, CA, 2001.

[Maes, 1994] Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, July 1994.

[Myers, 1985] Brad A. Myers. The importance of percent-done progress indicators for computer-human interfaces. In *Proceedings of CHI'85*, pages 11–17, 1985.

[Raskin, 2000] Jef Raskin. *The Humane Interface: New Directions for Designing Interactive Systems*. Addison Wesley, Reading, MA, 2000.

[Shneiderman and Maes, 1997] Ben Shneiderman and Pattie Maes. Debate: Direct manipulation vs. interface agents. *Interactions*, 4(6):42–61, November/December 1997.

[Shneiderman, 1998] Ben Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Addison-Wesley, 1998.

[Smith and Mosier, 1986] Sidney L. Smith and Jane N. Mosier. Guidelines for designing user interface software. Technical Report ESD-TR-86-278, The MITRE Corporation, Bedford, MA, 1986.

[St. Amant and Zettlemoyer, 2000] Robert St. Amant and Luke S. Zettlemoyer. The user interface as an agent environment. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 483–490, 2000.

[St. Amant, 1999] Robert St. Amant. User interface affordances in a planning representation. *Human Computer Interaction*, 14(3):317–354, 1999.

[Thimbleby, 2000] Harold Thimbleby. Calculators are needlessly bad. *International Journal of Human-Computer Studies*, 52(6):1031–1069, 2000.

Leveraging Data About Users in General in the Learning of Individual User Models

Anthony Jameson and **Frank Wittig***
Department of Computer Science, Saarland University
P.O. Box 15 11 50, 66041 Saarbrücken, Germany
jameson@dfki.de, wittig@cs.uni-sb.de

Abstract

Models of computer users that are learned on the basis of data can make use of two types of information: data about users in general and data about the current individual user. Focusing on user models that take the form of Bayesian networks, we compare four types of model that represent different ways of combining these two types of data. Models of the four types are applied to the data of an experiment, and they are evaluated according to theoretical, empirical, and practical criteria. One of the model types is a new variant of the AHUGIN method for adapting the probabilities of a Bayesian network while it is being used: *Differential adaptation* is a principled way of determining the speed with which each aspect of a network is adapted to an individual user.

1 Introduction

Machine learning techniques are being used increasingly in the development of interactive systems that adapt to individual users. Two contrasting approaches can be distinguished:

- *Learning general user models:* A system processes observations acquired from a sample of users so as to learn a model that applies to users in general (see, e.g., [Walker *et al.*, 2000]).
- *Learning individual user models:* While a user $\mathcal{U}$ is operating a system $\mathcal{S}$, $\mathcal{S}$ processes observations about $\mathcal{U}$ so as to learn a model of this particular $\mathcal{U}$ (see, e.g., [Segal and Kephart, 1999]).

Each of these types of learning has its own typical benefits, which will be discussed later in this paper. A natural strategy is to combine the advantages of general and individual models: Learn a general user model which can be applied to a new user; adapt the model to each user during the interaction.

One broad approach that applies this strategy is *collaborative filtering*, which has been applied widely in systems that recommend products (such as CDs) to users (see, e.g., [Herlocker *et al.*, 1999]). Here, the "general model" is essentially just the database of ratings (or other actions) that have been contributed by users so far. A system could predict how a given user $\mathcal{U}$ will rate a given object simply on the basis of this general model, by computing the average of all ratings given for that object. Instead, of course, collaborative filtering systems usually make individualized predictions that are based on the ratings of a subset of users who are especially similar to $\mathcal{U}$.

Although collaborative filtering systems have been highly successful, there are some application scenarios in which it is desirable to learn a more interpretable type of user model, in which causal relationships among variables are represented explicitly. For example, a system $\mathcal{S}$ may need to predict how the behavior of the user $\mathcal{U}$ will be influenced by particular contextual factors; or it may need to make uncertain inferences about unobserved contextual factors on the basis of $\mathcal{U}$'s behavior. Collaborative filtering is less straightforwardly applicable to this type of problem than another popular type of model: Bayesian networks (BNs).[1]

The main issue addressed in the present paper is: How can systems that employ Bayesian networks to model users most effectively exploit data about users in general and data about the current individual user?

Our investigation makes use of data from a controlled experiment.

2 Brief Description of Experiment

We begin by briefly summarizing the methods and results of the experiment (see [Müller *et al.*, 2001] for a more complete account). The experimental environment simulated on a computer workstation a situation in which a user is navigating through a crowded airport terminal while asking questions to a mobile assistance system via speech. In each trial, a picture appeared in a corner of the computer screen, and the subject was to introduce and ask a question related to the picture (e.g.,

*This research was supported by the German Science Foundation (DFG) in its Collaborative Research Center on Resource-Adaptive Cognitive Processes, SFB 378, Project B2 (READY). The experiment described in Section 2 was conducted in collaboration with Barbara Großmann-Hutter, Christian Müller, and Ralf Rummer. The suggestions of the three anonymous reviewers led to significant improvements. The first author is currently at DFKI, Saarbrücken.

[1] Explanations of Bayesian networks can be found in many sources, including the classic book by Pearl [1988]. An early survey of their application to user modeling was given by Jameson [1996].

"I'm getting thirsty. Will it be possible to get a beer on the plane?").

Two independent variables were manipulated orthogonally:

- TIME PRESSURE?: Whether the subject was instructed (a) to finish each utterance as quickly as possible or (b) to create an especially clear and comprehensible utterance, without regard to time.
- SECONDARY TASK?: Whether or not the subject was required to "navigate" through the terminal depicted on the screen by pressing arrow keys in order to move the cursor on the screen, avoiding obstacles in the process.

In each of the 4 (2×2) conditions, each of the 32 subjects produced 20 utterances. There are therefore 80 "observations" of each subject.

The subjects' speech input was later coded semi-automatically with respect to a wide range of features, including pauses, length, quality of content, and various types of disfluency. For the present study of learning methods, we selected a representative subset of four speech-related variables:

- ARTICULATION RATE: The number of syllables articulated per second of speaking time, not including silent pauses.
- NUMBER OF SYLLABLES: The number of syllables in the utterance.
- DISFLUENCIES: A binary variable that takes the value "True" when any one of four types of disfluency (e.g., starting a sentence but failing to complete it) is present in the utterance.
- SILENT PAUSES: The total duration of the silent pauses in the utterance, relative to the length of the utterance in words.

The practical relevance of this experiment lies mainly in the prospect that a mobile assistance system could interpret the features of a user's speech to make inferences about $\mathcal{U}$'s current psychological state ([Müller *et al.*, 2001]). In addition, there are situations in which it can be useful for $\mathcal{S}$ to be able to *predict* particular features of $\mathcal{U}$'s speech in a given situation—for example, so as to determine whether to request input via speech or via another modality.

3 Method and Model Types

The basic BN structure used for the models developed for the experiment is shown in the top two rows of Figure 1. We wish to simulate a situation in which a system $\mathcal{S}$ is interacting with a user $\mathcal{U}$ in this experimental situation and obtaining successive observations about $\mathcal{U}$. We will introduce four types of model, each of which will be tested according to the same procedure, which is shown in Table 1.

The characteristics of the four model types are summarized in Table 2; some further comments follow:

The *general model* is learned from the experimental data via the usual maximum-likelihood method for learning fully observable Bayesian networks (see, e.g., [Buntine, 1996]): The estimate of each (conditional) probability is computed simply in terms of the (relative) frequencies in the data. During the application to an individual user, the model is not adapted further: Essentially, a fresh copy of the model is used for the prediction of each observation.

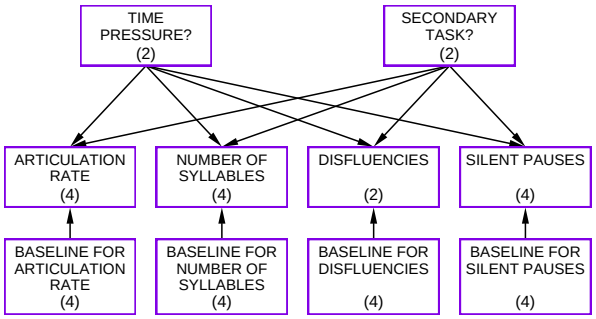


Figure 1. *Basic BN structure used for the comparison of models.*

(The four variables in the bottom row are included only in the parametrized model. The number in parentheses for each variable is the number of states of that variable.)

The *parametrized model* is initially learned in the same way as the general model, except that the four *parameter variables* shown in the bottom row of Figure 1 are included. Each of these variables represents the mean value of the corresponding variable above it for the user in question.

The *adaptive model* makes use of the AHUGIN ("adaptive HUGIN") method that was introduced by Olesen *et al.* [1992]. In contrast to the parametrized model, there is no explicit representation of the dimensions along which users may differ. Instead, the probabilities in the conditional probability tables (CPTs) of the BN are adapted whenever a new observation is processed. In this way, a great variety of individual differences can be adapted to, without any need for the designer of the BN to anticipate the nature of these differences. One question that arises in the application of the AHUGIN method concerns the speed with which the CPTs should adapt to the individual user. One simple approach, which is frequently used in other contexts, is to specify for the entire BN a single parameter, called the *equivalent sample size* (*ESS*); the ESS essentially represents the extent of the system's reliance on the initial general model, relative to the new data that will be obtained for each user. As we will see below, it is in general not obvious a priori what the most appropriate ESS for a given BN is. Moreover, the optimal value of the ESS can be quite different for different parts of the BN; in the context of our experiment, we may need a different ESS for each combination of a speech variable (e.g., NUMBER OF SYLLABLES) and an experimental condition. One original contribution of this paper is a principled method of estimating the optimal ESSs on the basis of the data collected with previous users. Application of this method yields a BN in which the various parts of the variables' CPTs adapt at different rates to each new user; hence the name *differential adaptation*. This method is described in detail in the Appendix.

The purely *individual model* is simply learned entirely on the basis of data from the current subject. So that some sort of inference can be performed right from the start, each CPT is initialized with uniform distributions. But as soon as the first observation for a given configuration of the values of the *parent variables* TIME PRESSURE? and SECONDARY TASK? has been

Initial model

- A BN defined in the way specified for type T (see Table 2) on the basis of the data from the other subjects in the experiment

Preparation of the test data

1. Determine a single random ordering of the 80 experimental stimuli, to be employed for all users
2. For each individual user $\mathcal{U}$, select the 80 observations for $\mathcal{U}$ according to this ordering

Testing the model for a single user $\mathcal{U}$ with respect to a variable V

For each observation O in the set of observations for $\mathcal{U}$,

1. Derive a belief about O:
   - Instantiate all variables for O other than V
   - Evaluate the BN to arrive at a belief regarding V
2. Determine the quadratic loss of the derived belief with respect to the actual value of V
3. Learn from the observation:
   - Use the values of all of the variables for the observation O to adapt the model, in the way specified for this type of model (see Table 2)

Presentation of results

- In each curve in a graph, the quadratic loss for each observation is aggregated over all subjects in the experiment
- Moreover, each value plotted is the mean quadratic loss for a block of 8 observations

Otherwise, sharp random fluctuations from one observation to the next would make it difficult to recognize general trends visually

Table 1. *Procedure for evaluating the learning (if any) and performance of a model of type* T.

Learning From Previous Users	Adaptation During Use by the Current User
General Model	
Learned on the basis of all observations of other subjects in the experiment, with no variables for individual parameters	No adaptation during use
Parametrized Model	
Learned on the basis of all observations of other subjects, with variables for individual parameters	The BN is built up as a dynamic BN, a new time slice being created for each observation; parameter nodes are updated as static nodes
Adaptive Model	
Individual models are learned for the other subjects as with the purely general model; on the basis of these models, an initial model for $\mathcal{U}$ is created that includes an equivalent sample size (ESS) for each configuration of values of parent variables in each conditional probability table (CPT)	After each observation, the CPT entries for the instantiated configurations of values of parent variables are updated according to the HUGIN algorithm, using the ESS computed for that configuration
Individual model	
No prior learning: In the initial model for $\mathcal{U}$, for each parent configuration in each CPT the probabilities are uniformly distributed and a minimal ESS of .0001 is specified	Adaptation is done as for the adaptive model, but the same minimal ESS is used for all parent configurations

Table 2. *Overview of the four model types compared.*

obtained, the initial model has essentially no further impact on the corresponding part of the CPT in question.

4 Results

We will now look at the results for these four models for each of the variables in the experiment. We first consider the four dependent variables in the second row of Figure 1, which need to be *predicted* by the model. Then we turn to the two independent variables in the top row. The derivation of a belief about one of these variables is essentially a *classification* task for the system: On the basis of an observation, the system attempts to assign $\mathcal{U}$ to a given experimental condition.

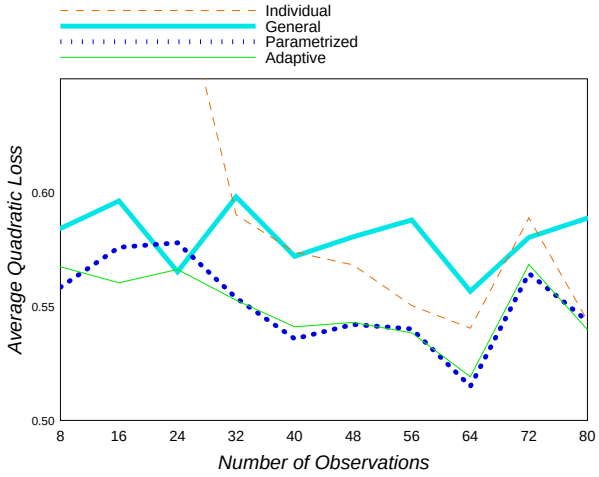


Figure 2. *Prediction accuracy for* ARTICULATION RATE. *(Higher values of* quadratic loss *represent lower accuracy.)*

4.1 Predicting a Variable With Simple Individual Differences

The results for the general model for the variable ARTICULATION RATE are shown by the solid thick curve in Figure 2. First, note that the only reason why this curve is not a straight horizontal line is that there is considerable random fluctuation in the quadratic loss variable; so there is no point in trying to interpret the individual zig-zags in the curves for the general model.

It is meaningful, on the other hand, to compare the overall performance of the general model with that of the parametrized and the adaptive models. As with all other comparisons that follow, we will use simple sign tests that take into account only the last 24 observations (3 blocks in the figures).[2] Each of these models performs consistently better than the general

[2] A test like this reflects how likely it is that a difference between the two curves in question would be found if we made the same comparison again, under the same circumstances and with the same subjects. It does not warrant a generalization to other subjects, tasks, models, etc.

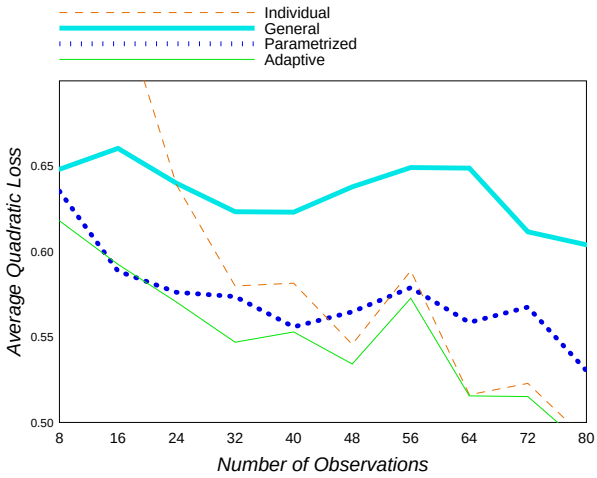


Figure 3. *Prediction accuracy for* NUMBER OF SYLLABLES.

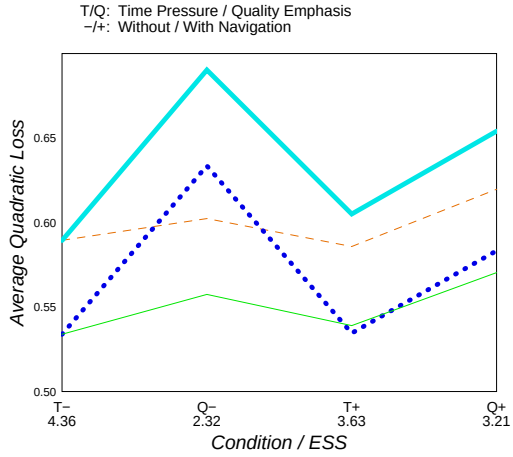


Figure 4. *Prediction accuracy for* NUMBER OF SYLLABLES *for each experimental condition.*

model during the last 24 observations ($p < .01$). This fact is understandable given that stable individual differences in articulation rate are known to exist. For this variable, the parametrized and the adaptive models perform very similarly.

The curve for the individual model shows a pattern that we will see to be typical: At first, the predictions are very inaccurate, as would be expected given that they are initially based on an arbitrary set of probabilities. But after about 30 observations the individual model does as well as the general model; and during the last 24 observations it is somewhat better ($p < .05$).

4.2 Predicting a Variable With More Complex Individual Differences

It is likewise generally known that people differ in their verbosity: the amount that they say in any given situation. Figure 3 shows the results for the variable NUMBER OF SYLLABLES. The individual differences are apparently even more important than for ARTICULATION RATE: The individual model catches up with the general model in the third block, and by the last three blocks it is tied for first place.

Moreover, the adaptive model significantly outperforms the parametrized model ($p < .02$ for the last 24 observations). Figure 4 helps to explain this advantage by displaying the accuracy levels for each of the four experimental conditions (without showing the time course of learning). The figure shows that the superior performance of the adaptive model occurs in just one of the four experimental conditions: Q−, in which subjects were instructed to produce high-quality utterances without having to navigate. Some subjects responded to this condition by creating elaborate, lengthy formulations, while others simply aimed to increase the clarity of an utterance of normal length. It is understandable that these individual differences should be hard to predict in terms of a single dimension of "verbosity", which is what the parametrized model has to use. The ability of the adaptive and individual models to learn $\mathcal{U}$'s behavior in each individual condition proves to be an advantage here. Note also that the ESS employed for the condition Q− is lower than that for the other conditions. In effect, the system has noticed that people respond very differently to this condition with respect to this variable and that it should therefore base its model largely on what it observes in the individual $\mathcal{U}$'s behavior.

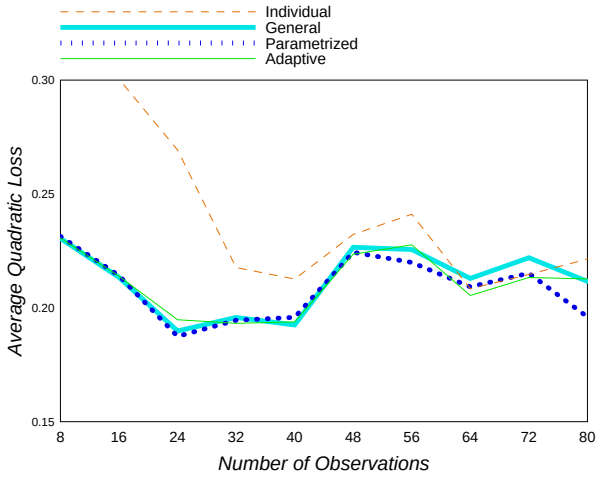


Figure 5. *Prediction accuracy for* DISFLUENCIES.

4.3 Predicting Low-Frequency Behaviors

The variable DISFLUENCIES (Figure 5) is an example of a variable for which there is little to be gained through adaptation to the individual user. On the average, only about 1 utterance in 8 contains one or more of the disfluencies in question. Consequently, it is inherently difficult for a system, given a limited number of observations, to acquire a model of a user's tendency to produce disfluencies which is better than the general model—even though stable differences among users might be found, given enough data. This fact is shown by the strong similarity of all four curves during the last few blocks of observations.

Measurable SILENT PAUSES (Figure 6) within an utterance are also rather infrequent events, occurring only about once in every 5 utterances overall. Still, the parametrized model does manage to outperform the general model here ($p = .02$).

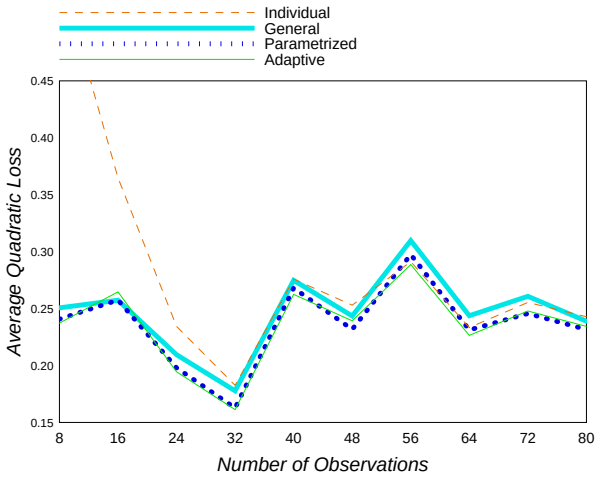


Figure 6. *Prediction accuracy for* SILENT PAUSES.

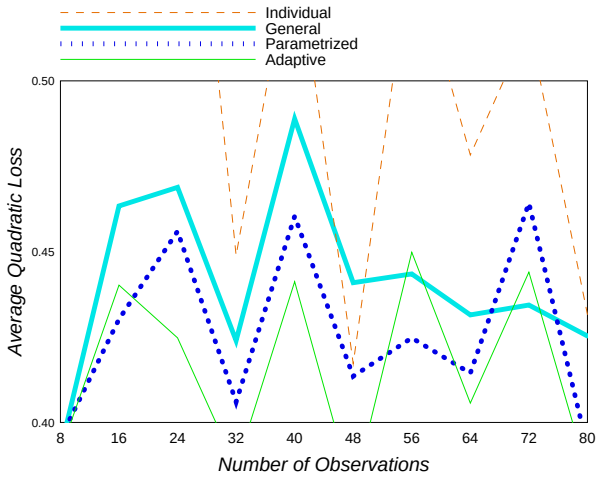


Figure 7. *Classification accuracy for* TIME PRESSURE?.

4.4 Inferring the Experimental Condition

Figure 7 shows the results for a classification task: Instead of predicting a particular aspect of $\mathcal{U}$'s behavior, $\mathcal{S}$ has to infer whether $\mathcal{U}$ was working under time pressure or with an emphasis on quality, given the features of one of $\mathcal{U}$'s utterances and knowledge of the value of the other independent variable (SECONDARY TASK?).

The pattern of relative accuracy shown in the figure is somewhat inconsistent with the pattern shown in the previous figures:

- The individual model never catches up with the other models in terms of accuracy, whereas it had done so for each of the prediction tasks.
- The parametrized and adaptive models do not show a very clear advantage over the general model ($p = .05$ and $p = .18$, respectively)—although one might have expected such an advantage, given that these models performed clearly better in the prediction of two of the four speech variables (ARTICULATION RATE and NUMBER OF SYLLABLES).

For classification with regard to SECONDARY TASK? (Figure 8), there are no reliable differences at all except that the individual model is much worse than the others throughout. This particular classification task—determining on the basis of a single utterance whether the subject is navigating or not—is very difficult, with performance being at best marginally above the chance level (which corresponds to a quadratic loss of 0.5).

These results remind us of the general point, argued by previous authors, that there is not necessarily one best model for a given set of data. For example, Friedman *et al.* [1997] discuss the reasons why a BN that is optimal with respect to some general accuracy criterion may perform suboptimally on classification tasks; and they propose methods for optimizing the classification accuracy of a learned BN. Greiner *et al.* [1997] have argued more generally that learning should take into account the specific queries that a BN is intended to answer.

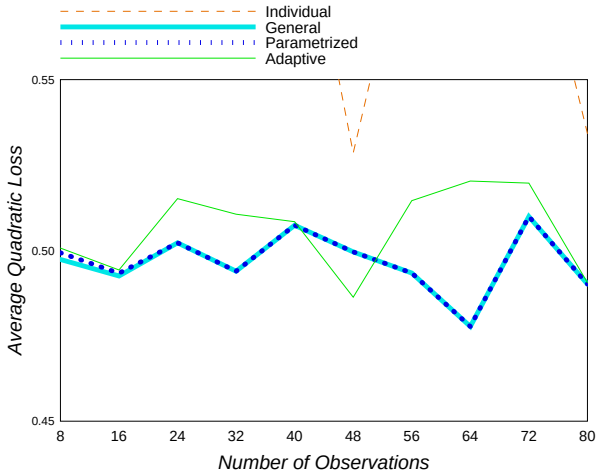


Figure 8. *Classification accuracy for* SECONDARY TASK?.

5 Discussion

5.1 Overall Comparison of Model Types

Table 3 summarizes the points that have been made in the preceding sections about (a) the theoretical strengths and weaknesses of the four model types, (b) the empirical results for them that were obtained in our experiment, and (c) practical considerations that may be equally important for the choice of a type of model.

The empirical results do not appear to depend on specific properties of this one experiment: When we performed the corresponding analyses for a quite different experiment (described in [Jameson *et al.*, 2001]), a very similar pattern emerged: In particular, for each of the categories of variables discussed in the subsections of the previous section, there was at least one variable in the other experiment that fell into that category and yielded similar results with regard to the performance of the four model types.

Regarding the empirical results: Although the parametrized and adaptive models perform best overall, the general and individual models each show competitive performance

Theoretical Considerations	Empirical Results	Practical Considerations
	General Model	
− Individual differences are never taken into account	− Out-performed in the long run by the parametrized and adaptive models, except where individual differences are small or very hard to learn; and sometimes by the individual model	− Ample prior data required + No overhead for run-time adaptation
	Parametrized Model	
+ Knowledge about the nature of individual differences can be represented explicitly − Many parameters may be required if individual differences are complex	+ Generally better than the general and individual models and competitive with the adaptive model − Somewhat poorer than the adaptive or even the individual model when individual differences are complex	− Ample prior data required − Dynamic Bayesian networks can raise complexity problems + Parameters can be shared with other user models
	Adaptive Model	
+ Specific parts of the model are adapted at different rates in a principled manner + Permits smooth transition from a general model to an individual model − The number of degrees of freedom for the learning may be unnecessarily high, relative to the paramaterized model, so that learning is unnecessarily slow	+ Generally good performance, especially on prediction tasks with complex individual differences	− Ample prior data required + No prior *explicit* knowledge about individual differences required − The adaptation mechanism must be invoked repeatedly during system use
	Individual model	
− Since no use is made of prior knowledge or data, inference is likely to be very inaccurate during the initial phase of use + There is no bias against entirely unexpected patterns of behavior	− Very poor performance during some initial phase of use − The phase of poor performance is especially long for classification tasks + Good ultimate prediction performance where behavior is highly idiosyncratic	+ No prior knowledge or data required − The adaptation mechanism must be invoked repeatedly during system use

Table 3. *Overview of the strengths and limitations of the four model types.*

under certain conditions. Consequently, one of these models may be turn out to be the most suitable one when these conditions are met and the practical considerations favor the model in question.

The use of Table 3 to select a model type for a given application scenario is made more difficult by the fact that some of the conditions mentioned (e.g. "individual differences are complex") refer to properties of the data that may or may not be known a priori. It may therefore be necessary to test two or more types of model empirically on data from the domain in question before arriving at a decision. Even in these cases, Table 3 should be helpful in that it calls attention to the key considerations and the most promising model types.

5.2 Novel Aspects of the Differential Adaptation Method

The most salient features of the method of differential adaptation are the following:

1. It leverages data about previous users not only to learn an initial general user model but also to learn how fast the various aspects of this model should adapt to each new individual user.
2. It does so without requiring the explicit representation of dimensions along which individual users may differ, as is the case with parametrized models.

As another example of a scenario in which this method might be useful, consider the system Syskill & Webert ([Pazzani and Billsus, 1997]), which predicts whether a given web page will be interesting to the current user. In the main version of the system, the user model consists essentially of a set of probabilities for each word W in a set of relevant words: $p(W \textit{ is present} | \textit{page is interesting})$ and $p(W \textit{ is present} | \textit{page is uninteresting})$. Pazzani and Billsus note that it can take inconveniently long for the system to learn the necessary probabilities solely on the basis of $\mathcal{U}$'s page ratings, and they propose solutions to this problem. In accordance with the basic idea of differential adaptation, this problem could be dealt with as follows: For each word W, the system would derive a prior expectation (more precisely, a beta distribution), for $p(W \textit{ is present} | \textit{page is interesting})$ on the basis of the data of other users, through the procedure described in the Appendix, along with a corresponding expectation for $p(W \textit{ is present} | \textit{page is uninteresting})$; and these expectations would be used and updated as in the tests described above. This method might be especially useful in the case of words that (a) occur infrequently but (b) exhibit similar probabilities for most users.

6 Summary of Contributions

As the learning of user models for adaptive systems becomes more widespread, increasing attention will be devoted to the

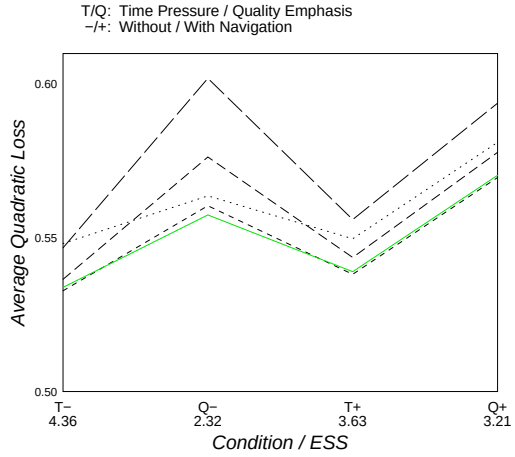


Figure 9. *Comparison between the differential adaptation method and the use of fixed ESSs.*

(The solid line shows the results for the adaptive model which were shown in Figure 4; The dashed lines show the results for models using fixed ESSs of 1, 5, 10, and 20, respectively, with the length of the dashes reflecting the ESSs.)

goal of making optimal use of the available data. Focusing on models that take the form of Bayesian networks, this paper has (a) systematically compared, with regard to several criteria, four representative ways of exploiting data about users in general and/or individual users; and (b) introduced a variant of the AHUGIN adaptation method called *differential adaptation*, which represents a principled way of determining the speed with which the various aspects of a general model should be adapted to an individual user.

A Appendix: The Differential Adaptation Method

As was noted in Section 3, the simplest type of adaptive model that can be realized with AHUGIN is one in which a single equivalent sample size (ESS) is specified for an entire BN. Figure 9 illustrates the limitations of this method: The solid line, which is repeated from Figure 4, shows the predictive accuracy of the adaptive model that used the method of differential adaptation; recall that this model computed and used the four ESSs shown under the x-axis, one for each experimental condition for the variable NUMBER OF SYLLABLES. Each of the dashed lines shows the results for a model that used a constant ESS of 1, 5, 10, or 20, respectively. The results for the ESSs of 1, 10, and 20 are noticeably worse than those for the ESS of 5 and for differential adaptation; that is, the choice of an ESS really does make a difference. The fact that the accuracy for the ESS of 5 is only slightly worse than that for differential adaptation is not surprising, given that the ESSs computed by differential adaptation are fairly close to 5. The main contribution of the differential adaptation method here is to compute the right general level of the ESSs automatically, avoiding the need for trial and error on the part of the designer of the BN. Differential adaptation also gains a bit of additional accuracy by computing a different ESS for each experimental condition, in particular choosing a lower value for the condition "Q–", in which individual differences are especially large (cf. 4.2).

Similarly, for each of the other variables examined in this experiment and the experiment mentioned in 5.1, differential adaptation consistently performed at least as well as the best fixed-ESS model, deriving ESSs ranging from 12.9 to essentially 0. The differences from the fixed-ESS models were in most cases smaller than those shown in Figure 9. Still, since differential adaptation is computationally quite straightforward (see below), there appears to be no reason not to use it whenever it is applicable.

The rest of this Appendix explains in more detail the differential adaptation method and the relevant aspects of the AHUGIN adaptation facility.[3]

Suppose that a variable X has K possible states. To simplify notation, all mathematical expressions that follow refer to one particular configuration c of states of the parent variables of X. For this configuration, there are K probabilities p_k in the conditional probability table (CPT) for X. In addition to storing these K probabilities, AHUGIN maintains for each c a Dirichlet distribution (see, e.g., [Heckerman, 1995, Section 2]; [Olesen *et al.*, 1992]) that represents the system's current expectation concerning the true vector of probabilities of which p_k are simply the current estimates. The parameters of each such Dirichlet distribution are as follows:

- a vector of K means m_k;
- an *equivalent sample size* (ESS, denoted in the formulas as s).

The means m_k are numerically identical to the K CPT entries p_k, but it will be helpful to denote them with m_k when we are viewing them as parameters of the Dirichlet distribution.

Whenever a new observation is obtained in which the configuration c of states of X's parents is realized, s is incremented by 1 and the m_k are updated according to the usual method for Dirichlet distributions.

In the context of learning user models, the AHUGIN algorithm gives us the opportunity to specify a Dirichlet distribution for each configuration c of states of parent variables of each CPT in a BN for a new user $\mathcal{U}$. This opportunity can be exploited as follows if we have complete data from N other users:

1. Learn N separate BNs from the data, one for each previous user, using the standard maximum likelihood method for learning fully observable BNs with known structure (see, e.g., [Buntine, 1996]).

2. For each configuration c of states of parent variables of a variable X, each learned BN_n yields a vector of empirically determined conditional probabilities p_{nk}. These N vectors can be viewed as a sample of vectors upon which we can base our expectation concerning the corresponding vector that we will obtain for a new user after collecting a lot of data on that user.

The question now is how we can represent this expectation as an initial Dirichlet distribution with K dimensions, as is re-

[3] See Olesen *et al.* [1992] for a much more complete description of AHUGIN, which is now available as part of the HUGIN software package (see http://www.hugin.com).

quired by the AHUGIN method. Olesen *et al.* [1992] describe a straightforward method for specifying a Dirichlet distribution that comes close to matching a given distribution that is specified in another way. First, the K means of the Dirichlet distribution should match the means of the original distribution exactly. In our case, this implies that each initial mean m_k should be defined as follows:

$$m_k = \frac{\sum_{n=1}^{N} p_{nk}}{N}. \quad (1)$$

That is, each m_k is simply the mean of the N CPT entries from the existing BNs.

Ideally, each of the K variances v_k of the Dirichlet distribution should match the variance of the corresponding N existing CPT entries. In general this goal will be unattainable, since there is only one degree of freedom available for determining the variance of the Dirichlet distribution, namely the ESS s. Olesen *et al.* [1992] propose choosing the ESS so that the (weighted) *average* of the variances of the Dirichlet distribution (denoted with v) equals the weighted average of the variances of the original distribution. Given the formula for the variance of one dimension of a Dirichlet distribution,

$$v_k = \frac{m_k(1-m_k)}{s+1}, \quad (2)$$

we have the following formula for the weighted average variance:

$$v = \frac{\sum_{k=1}^{K} m_k^2(1-m_k)}{s+1}. \quad (3)$$

Solving for s, we obtain:

$$s = \frac{\sum_{k=1}^{K} m_k^2(1-m_k)}{v} - 1. \quad (4)$$

To obtain the appropriate ESS, we need only to replace v in this formula with v', the computed average of the K variances in the empirically obtained CPTs. Each of these K variances v'_k is given by the formula

$$v'_k = \frac{\sum_{n=1}^{N} (p_{nk} - m_k)^2}{N}, \quad (5)$$

since m_k has already been computed as the mean of the corresponding p_{nk}.

To compute the weighted average variance, we can likewise use as weights the values m_k:

$$v' = \sum_{k=1}^{K} m_k v'_k. \quad (6)$$

Putting it all together, the most appropriate ESS can be computed directly from the original CPT entries p_{nk} and the corresponding means m_k as follows:

$$s = \frac{N \sum_{k=1}^{K} m_k^2(1-m_k)}{\sum_{k=1}^{K} m_k \sum_{n=1}^{N} (p_{nk} - m_k)^2} - 1. \quad (7)$$

References

[Buntine, 1996] Wray Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, 8:195–210, 1996.

[Friedman *et al.*, 1997] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

[Greiner *et al.*, 1997] Russell Greiner, Adam J. Grove, and Dale Schuurmans. Learning Bayesian nets that perform well. In Dan Geiger and Prakash P. Shenoy, editors, *Uncertainty in Artificial Intelligence: Proceedings of the Thirteenth Conference*, pages 198–207. Morgan Kaufmann, San Francisco, 1997.

[Heckerman, 1995] David Heckerman. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1995. Revised November 1996.

[Herlocker *et al.*, 1999] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 1999 Conference on Research and Development in Information Retrieval*, 1999.

[Jameson *et al.*, 2001] Anthony Jameson, Barbara Großmann-Hutter, Leonie March, Ralf Rummer, Thorsten Bohnenberger, and Frank Wittig. When actions have consequences: Empirically based decision making for intelligent user interfaces. *Knowledge-Based Systems*, 14:75–92, 2001.

[Jameson, 1996] Anthony Jameson. Numerical uncertainty management in user and student modeling: An overview of systems and issues. *User Modeling and User-Adapted Interaction*, 5:193–251, 1996.

[Müller *et al.*, 2001] Christian Müller, Barbara Großmann-Hutter, Anthony Jameson, Ralf Rummer, and Frank Wittig. Recognizing time pressure and cognitive load on the basis of speech: An experimental study. In Julita Vassileva and Piotr Gmytrasiewicz, editors, *UM2001, User Modeling: Proceedings of the Eighth International Conference*. Springer, Berlin, 2001.

[Olesen *et al.*, 1992] Kristian G. Olesen, Steffen L. Lauritzen, and Finn V. Jensen. aHUGIN: A system creating adaptive causal probabilistic networks. In Didier Dubois, Michael P. Wellman, Bruce D'Ambrosio, and Phillipe Smets, editors, *Uncertainty in Artificial Intelligence: Proceedings of the Eighth Conference*, pages 223–229. Morgan Kaufmann, San Mateo, 1992.

[Pazzani and Billsus, 1997] Michael Pazzani and Daniel Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27:313–331, 1997.

[Pearl, 1988] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[Segal and Kephart, 1999] Richard B. Segal and Jeffrey O. Kephart. MailCat: An intelligent assistant for organizing e-mail. In *Proceedings of the Third International Conference on Autonomous Agents*, pages 276–282, 1999.

[Walker *et al.*, 2000] Marilyn A. Walker, Irene Langkilde, Jerry Wright, Allen Gorin, and Diane J. Litman. Learning to predict problematic situations in a spoken dialogue system: Experiments with How May I Help You? In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL'00)*, pages 210–217, Seattle, WA, 2000.

MULTI-AGENT SYSTEMS

MULTI-AGENT SYSTEMS APPLICATIONS

An Agent Architecture for Multi-Attribute Negotiation

Catholijn M. Jonker and Jan Treur
Vrije Universiteit Amsterdam, Department of Artificial Intelligence
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
Email: {jonker, treur}@cs.vu.nl URL: http://www.cs.vu.nl/{~jonker,~treur}

Abstract

A component-based generic agent architecture for multi-attribute (integrative) negotiation is introduced and its application is described in a prototype system for negotiation about cars, developed in co-operation with, among others, Dutch Telecom KPN. The approach can be characterised as co-operative one-to-one multi-criteria negotiation in which the privacy of both parties is protected as much as possible.

1 Introduction

In [Gutman and Maes, 1998] the difference between competitive and co-operative negotiation is discussed. Guttman and Maes state that the competitive negotiations in retail markets are unnecessarily hostile to customers and offer no long-term benefits to merchants. Essentially, in competitive negotiations the merchant is pitted against the customer in price-tug-of-wars. Based on [Forrester, 1997], in [Gutman and Maes, 1998] it is concluded that merchants often care less about profit on any given transaction and care more about long-term profitability, which implies customer satisfaction and long-term customer relationships. Their analysis makes a strong case for co-operative negotiation for the retail market:

> "...the multi-attribute utility theory (MAUT) [Keeny and Raifa, 1976], can help customers make complex buying decisions taking into account multiple factors including merchants' unique added value (e.g., extended warranty options, delivery options, etc.)."

Their argument is supported by [Rosenschein and Zlotkin, 1994], which makes clear that co-operative negotiation can be described as a non-zero-sum game where, as the values along multiple dimensions shift in different directions, it is possible for all parties to be better of. Thus co-operative negotiation is a win-win type of negotiation.

The Consumer Buying Behaviour Model (CBB) (see [Gutman and Maes, 1998]) consists of six main stages: Need Identification, Product Brokering, Merchant Brokering, Negotiation, Purchase and Delivery, and Service and Evaluation. The model discussed in this paper addresses the first four of these stages, where the product brokering is an integrated part of the entire brokering process and overlaps with the need identification. This is in line with normal procedures, as "CBB stages often overlap and migration from one to another is sometimes non-linear and iterative". The buyer contacts the broker agent, the broker agent provides the buyer with forms in order to determine the wishes of the buyer. Then the broker matches products and suppliers against the wishes of the buyer presenting him with the best three options. The buyer can then select one of these proposals. A special buyer representative agent negotiates with the representative agent of the supplier to obtain the best configuration of the selected option. The different attributes of the object under negotiation, the possible values for each of those attributes, and the different wishes (profiles) of consumer and provider, allow for co-operative negotiation: co-operative negotiation can be seen as a decision-making process of resolving a conflict involving two or more parties over multiple interdependent, but non-mutually exclusive goals; cf. [Lewicki et al., 1997].

The multi-agent system in which the negotiation agent can be and has been applied consists of the following types of agents: Human Buyers, Human Dealers, Buyer Representative agents, Dealer Representative agents, Broker agent. Moreover, to model retrieval of information from databases, a number of components is used; one of them is the External World from which Buyer Representative agents can retrieve third party information (consumer organisations like the AA of the US and the Dutch ANWB). Furthermore, specific Dealer-dependent Dealer Databases for all Dealers are included, from which the Dealer Representative agent can retrieve information about the cars offered by that particular Dealer. Because of space limitation, this paper focuses on the negotiation process within this overall architecture. The generic agent architecture for multi-attribute negotiation was designed and formally specified using DESIRE, as a refinement of the Generic Agent Model GAM [Brazier et al., 2000].

In this paper, in Section 2 the most sophisticated component within the agent architecture, Cooperation

Management, which models the negotiation process, is described in more detail. In Section 3 the prototype system developed on the basis of the agent architecture is discussed; example results are shown. Section 4 concludes the paper by a discussion.

2 The Negotiation Model

In multi-attribute negotiation a bid has the form of values assigned to a number of attributes. For example, if the negotiation is about cars, and the relevant attributes are cd player, extra speakers, airco, tow hedge, price, then a bid consists of an indication of which CD player is meant, which extra speakers, airco and tow hedge, and what the price of the bid is. In the current section the *generic* negotiation model is described; for instantiations, see Section 3.

To assess a bid of the other party, it is important to have evaluation methods. Evaluation can be done at two levels: the level of each of the specific attributes (attribute evaluation), and the level of the bid as a whole (overall bid utility). Taking this into account, some characteristics of the multi-attribute negotiation model presented here are:

- explicit reasoning about the negotiation strategy and co-ordination of the negotiation process
- evaluation of a bid takes into account both the attributes separately and the overall utility of the bid
- planning of a new bid takes into account both the overall utility level and the level of attributes separately

In particular, in the model it is possible to work on two levels: the level of the overall bid, and the level of each of the attributes separately. The negotiation model has been specified as a compositional structure within the component Cooperation Management of GAM [Brazier et al., 2000]. Globally speaking, the process runs as follows:

- For each negotiation round, first evaluations of the attributes of the previous bids are determined.
- Then these evaluations are aggregated into overall utilities of these previous bids.
- Next, it is determined which concession step is made for the next bid, expressed in terms of the overall utility; this provides a target utility.
- To obtain the next bid, given the target utility, first according to some distribution over attributes, target attribute evaluation values are determined (chosen in such a manner that they aggregate exactly to the target utility)
- Finally, for each of these target attribute evaluation values, an attribute value is chosen that has an evaluation value as close as possible to the target evaluation value for the attribute.

In the last step, if only discrete attribute values exist, it may be the case that the target utility is not reached. However, if at least one of the attributes has continuous values, then this attribute can be chosen to compensate for differences that are created due to the mapping to discrete values for the other attributes. In our application, the price attribute is such a continuous attribute, and chosen to compensate for differences. In this manner bids are created that exactly match the target utilities. To realise the compositional process structure sketched above, at its top level the component Cooperation Management is composed of the five components in Figure 1: Negotiation Coordination, Attribute Evaluation, Bid Utility Determination, Utility Planning, and Attribute Planning. Each of these components is discussed in more detail.

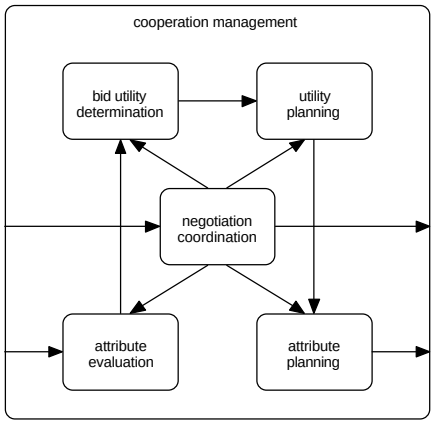


Figure 1 The Multi-Attribute Negotiation Model

2.1 Negotiation Coordination

Within the component Negotiation Coordination the negotiation process state is analysed (component Process Analysis) and the process is controlled (component Process Control). Process Analysis determines which of the following are true and which are false:

a) *Repetition of steps* takes place: steps without enough progress (depending on the impatience factor (π) which specifies the acceptable number of steps in which nothing changes)

b) A *utility gap* (larger than some threshold ω) remains; i.e., a significant difference between the utility of the own bid and that of the other agent's bid.

c) A *configuration mismatch* (larger than some threshold ν) remains between the own bid and the other agent's bid.

Here a configuration mismatch means that for at least one attribute, between the two values (in the two bids) a significant difference exists. Depending on the outcome of

the analysis within component Process Control the following actions can be decided upon:

1. Start a *next negotiation round*
2. Contact the user to discuss whether the *concession factor* (γ) can be changed.
3. Contact the user to discuss whether the *configuration tolerance* (τ) can be changed.
4. Communicate to the user that an *agreement* has been reached.
5. Communicate to the user that the *negotiation has failed* (only when the user is unwilling to change the characteristics).

Action	*Repetition*	*Utility gap*	*Config. Match*
next round	No		No
discuss concession factor	Yes	Yes	No
discuss config. tolerance	Yes	No	No
report success		No	Yes
report failure	Yes		No

Table 1 Action decision table

2.2 Attribute Evaluation

Evaluation of the attributes is made on the basis of the evaluation functions that are part of the user profile maintained within component Maintenance of Agent Information of the agent. Component Attribute Evaluation evaluates the attributes of available objects based on the preferences of the user represented by the agent.

The evaluation functions either have a *table form* or another *specific function description*. A table form is used for discrete attributes such as accessories. Specific function descriptions are used for continuous attributes such as mileage or price. The form of specific function descriptions are of a type such as 'linear', or 'uphill'. For the attributes for which a specific (non-table) type of evaluation function is given, depending on this type, knowledge is specified to obtain the object evaluations. Currently, only specific function types are used that consist of linear parts, cut off between 0 and 1: linear function, normal distribution function, downhill function, uphill function.

If desired, in all evaluations and utilities, the model supports that two aspects can be modelled separately and integrated: *ease evaluation* and *ease utility* EU and *financial evaluation* and *financial utility* FU. The latter aspect covers the financial rationality in the agent's behaviour. The former aspect models all other aspects within the decision making such as a resistance against more complicated transactions (even if in terms of economic gain they are more favourable). The balance between these two aspects within the overall evaluations is defined by the *financial rationality factor* ρ. If this factor is 1, then only the financial utility is taken into account (completely financially driven), if it is 0, only the ease utility (completely ease driven). Any factor in between 0 and 1 defines the relative weight of the economic aspect compared to the ease aspect in the decision making. For example, for a certain accessory the financial aspect of the evaluation value is the cost it takes to provide it (both the price of the accessory and the cost of installing it).

2.3 Bid Utility Determination

Within the model, the *utility* U_B of a bid B is taken as a weighted sum of the *attribute evaluation values* $E_{B,j}$ for the different negotiant attributes denoted by j.

$$U_B = \Sigma_j \, w_j \, E_{B,j}$$

Here the weights w_j are relative importance factors based on the *importance factors* p_k for the different attributes:

$$w_j = p_j / \Sigma_k \, p_k$$

If a financial utility is used separately, then the above utility (called the *ease utility* EU_B) is determined on the basis of all attributes except price. *Financial utility* FU_B is based on the financial balance g_B for a given bid B:

$$g_B = p_B - b - a_B$$

where b denotes the *basic costs* (the cost of the object without additional accessories) and a_B denotes the *additional costs* of bid B, and p_B price within bid B.

$$a_B = \Sigma_j \, FE_{B,j}$$

that is based on the financial evaluations ($FE_{B,j}$) of the values of the different attributes j. However, to be able to relate FU_B to the ease utility EU_B, FU_B is the normalisation of the financial balance to a number between 0 and 1:

$$FU_{B,j} = g_B / \delta b$$

The fraction δ is the fraction of the basic cost that is maximally additionally (to be) earned (e.g. 0.3, a *maximum margin* of 30%). Some notes can be made.

- The financial utility FU is defined on the interval between 0 and 1 in such a manner that financial utility 1 means cost price plus maximal margin ($b + \delta b + a_B$).
- Let B_0 be the initial bid of the seller, then by taking price $p_{B0} = b + \delta b + a_{B0}$, the financial utility of this bid is

  $$FU_{B0} = (p_{B0} - b - a_{B0}) / \delta b = 1.$$

- By setting δ properly, the seller makes sure that (s)he is not asking unrealistic prices.
- The financial utility is defined on the interval between 0 and 1 in such a manner that $FU_B = 0$ implies $p_B = b + a_B$, i.e., the cost price. So, if a buyer makes a bid B with $p_B < b + a_B$, then $FU_B < 0$ from the perspective of the seller.

On the basis of the ease utility and the financial utility, the *overall utility* is determined as a weighted sum. Here the weights are based on the financial rationality factor ρ (part of the dealer profile).

$$U_B = \rho FU_B + (1 - \rho) EU_B$$

2.4 Utility Planning

For determination of the *target utility* TU the following formula is used within the model:

$$TU = U_{BS} + CS$$

with U_{BS} the utility of the own bid, and the concession step CS determined by

$$CS = \beta (1 - \mu / U_{BS}) (U_{BO} - U_{BS})$$

where U_{BO} is the utility of the other agent's bid. In this formula the factor $U_{BO} - U_{BS}$ expresses the current *utility gap*. The factor $(1 - \mu / U_{BS})$ expresses that the concession step will decrease to 0 if the U_{BS} approximates the minimal utility μ. This ensures $U_{BS} \geq \mu$. The factor β stands for the *negotiation speed*. The minimal utility is taken as $\mu = 1 - \gamma$ with γ the *concession factor*, expressing a measure in how far concessions can be made. Determination of the target utility can also address the ease and financial aspect separately (indicated by E or F added to the parameters). For each of these aspects the same model is used. For example, for the ease aspect the following formula is used:

$$TEU = EU_{BS} + ECS, \quad \text{with}$$

$$ECS = \beta_E (1 - \mu_E / EU_{BS}) (EU_{BO} - EU_{BS})$$

In this formula β_E is the negotiation speed factor for the ease part, and μ_E is the minimal ease utility. Similarly, for the financial aspect the target utility is:

$$TFU = FU_{BS} + FCS, \text{ with}$$

$$FCS = \beta_F (1 - \mu_F / FU_{BS}) (FU_{BO} - FU_{BS})$$

The speed factors β_E for ease and β_F for financial parts are based on the negotiation speed factor β and the financial rationality factor ρ as follows

$$\beta_E = (1 - \rho) \beta \qquad \beta_F = \rho \beta$$

The *minimal ease utility* is taken as $\mu_E = 1 - \gamma$. The *minimal financial ut*ility is taken as $\mu_F = \varepsilon / \delta$ where ε is the *minimal financial margin*. The explanation is as follows. If the minimal margin is achieved, then the price $minP$ is

$$minP = \varepsilon b + b + a_B$$

Given $minP$, the minimal acceptable financial utility can be calculated as follows:

$$\mu_F = (minP - b - a_B) / \delta b = \varepsilon b / \delta b = \varepsilon / \delta$$

For example, if $\delta = 0.2$ (20%) and $\varepsilon = 0.1$ (10%), then $\mu_F = 0.5$, i.e., the dealer is not willing to sell with a financial utility lower than half of its maximal financial utility (based on the maximal margin); a financial utility of 0 means selling against the cost price, i.e., no margin at all, a financial utility of 1 means selling with a margin of 20% on the cost price.

2.5 Attribute Planning

The Attribute Planning process uses as input the target utility and determines as output the configuration for the next (own) bid in the following two main steps:

- First, within the component Target Evaluation Determination, for each attribute a target evaluation is determined.
- Next, given these target evaluations per attribute, within the component Configuration Determination, a configuration for the next bid is determined.

Target Evaluation Determination

Target evaluations per attribute TE_j are determined in the model in two steps. First a *basic target evaluation* per attribute BTE_j is determined in such a way that $\Sigma_j w_j BTE_j = TU$. Then the *target evaluations* TE_j are combinations of the BTE_j with the evaluations of the attributes in the bid of the negotiation partner. The basic target evaluation per attribute BTE_j is determined according to the following format:

$$BTE_j = E_{BS,j} + (\alpha_j / N) (TU - U_{BS})$$

Here the α_j can be chosen arbitrarily, and N is a normalisation factor. Factor N is defined as the weighted sum of the α's with the relative importance factors as weights: $N = \Sigma_j w_j \alpha_j$. Due to this normalisation factor, the utility determined as a combination of the target evaluations leads to exactly the target utility:

$$\begin{aligned}\Sigma_j w_j BTE_j &= \Sigma_j w_j (E_{BS,j} + (\alpha_j / N) (TU - U_{BS})) \\ &= \Sigma_j w_j E_{BS,j} + \Sigma_j w_j (\alpha_j / N) (TU - U_{BS}) \\ &= U_{BS} + 1/N \Sigma_j w_j \alpha_j (TU - U_{BS}) \\ &= U_{BS} + 1/N * N * (TU - U_{BS}) \\ &= TU\end{aligned}$$

The choice for the α's is made as: $\alpha_j = (1 - w_j) (1 - E_{BS,j})$. The first factor expresses the influence of the user's own importance factors (similar to the choice made in [Benn, et al., 1999]); the second factor takes care that the target evaluation values remain scaled in the interval between 0 and 1. Besides the influence on the target attribute evaluations as described, also a concession to the opponent's attribute evaluations is made. This depends on the configuration tolerance τ, as follows:

$$TE_j = (1 - \tau) BTE_j + \tau E_{BO,j}$$

If the configuration tolerance is 0, then only the user's importance factors are taken into account. If the configuration tolerance is 1, then with respect to the configuration maximal concession to the negotiation partner is made.

Configuration Determination
To determine a configuration for the next bid the following three steps are made.

- First, for each attribute, given the target evaluation, attribute values are determined with an evaluation that is as close as possible to the target evaluation value.
- Next, a partial configuration (price attribute not yet filled) is determined based on these closest values.
- Finally, to complete the configuration for the next bid, also the price attribute value is determined.

The partial configuration is selected from the closest attribute values. If more than one choice with closest value is possible, then, if it is among the options, the value in the opponent's bid is chosen, otherwise the choice is made in a random manner. The partial configuration is completed by determining the price attribute value in such a manner that the overall target utility is achieved.

Within the Dealer Representative agent a simple possibility would be to take the target financial utility as the aim to be achieved. However, due to the discrete values of the accessory attributes, the ease utility will probably not be exactly achieved. The choice has been made that this difference is compensated in the financial utility. For example, if the ease utility of the partial configuration is lower than the target ease utility, then the Dealer Representative agent aims at a financial utility which is (in proportion) higher than the target financial utility.

First the ease utility of the partial configuration is determined. Next the financial utility that has to be achieved (AFU) is determined, as the (weighted) difference between overall target utility and the realised ease utility:

$$AFU = TU - (1 - \rho)\, U_{P,E} / \rho$$

where $U_{P,E}$ is the ease utility of the partial configuration P. Finally, the price attribute value is determined, as the sum of all costs and the fraction of the maximum margin given by the financial utility aimed for:

$$price = b + a_P + AFU\ \delta\ b$$

3 Implementation: Example

A trace was generated on the basis of the data presented in the Tables 2 to 5. In these tables, "buyer representative" is abbreviated to BR, similarly DR stands for dealer representative. Basic negotiation parameters are depicted in Table 2 above. The buyer representative only uses the (ease) evaluations and utilities; therefore, the special financial factors are not applicable in Table 2. In Table 3 the importance factors are depicted. In Table 3, for the dealer representative the price attribute has no value, since it is only part of the financial utility function. Within financial terms the importance factors are irrelevant.

Negotiation parameter	BR	DR
negotiation speed β	0.5	0.4
impatience factor π	4	4
configuration gap size in price ν	250	200
utility gap size ω	0.02	0.02
concession factor γ	0.5	0.9
configuration tolerance τ	0.5	0.9
financial rationality factor ρ	not applicable	0.5
minimal financial margin ε	not applicable	0.1
maximal financial margin δ	not applicable	0.3

Table 2 Negotiation parameters in the example

(Ease) Importance factor	BR	DR
cd	0.8	0.6
extra speakers	0.8	0.2
airco	0.2	0.2
tow hedge	0.3	0.9
price	0.5	not applicable

Table 3 Importance factors ρ_k

In Table 4 below the evalution descriptions for the different attributes are depicted.

Evaluation description for cd player	BR	DR
good	1	0.58
fairly good	0.8	0.6
standard	0.75	0.7
meager	0.7	0.3
none	0	0.65
Evaluation description for extra speakers	**BR**	**DR**
good	1	0.2
fairly good	0.95	0.8
standard	0.9	0.9
meager	0.2	0.2
none	0	0.85
Evaluation description for airconditioning	**BR**	**DR**
good	0.97	0.9
fairly good	0.98	0.85
standard	0.99	0.2
meager	1	0.2
none	0	0.89
Evaluation description for tow hedge	**BR**	**DR**
good	0.97	0.6
fairly good	0.98	0.7
standard	0.99	1
meager	1	0.2
none	0	0.65
Evaluation description for price	**BR**	**DR**
function form	downhill	not applicable
critical value	16000	not applicable
steepness	-0.00015	not applicable

Table 4 Evaluation descriptions for the attributes

Furthermore, the dealer representative also needs the financial evaluation descriptions for the different accessories and it needs to know the basic costs of the car under negotiation (in this case the basic costs are 13000). The dealer representative's financial evaluation descriptions are depicted in Table 5 below.

Accessory	good	fairly good	standard	meager	none
cd	700	600	500	300	0
extra speakers	500	400	300	200	0
airco	2000	1700	1500	1200	0
tow hedge	500	400	300	200	0

Table 5 Financial evaluation descriptions for dealer

A trace of the negotiation process is depicted in the Tables 6 and 7 below. The buyer representative's bid, his opinion of his bid and his opinion of the bid of the dealer representative in the previous round are presented in Table 6 below.

BR	round 1	2	3	4	5	closing:12
bid						
price	16000	16979	17595	17765	18187	18723
tow hedge	meager	meager	meager	meager	meager	meager
airco	meager	meager	meager	meager	meager	meager
extra speakers	good	good	good	good	good	good
cd player	good	good	good	meager	good	good
utility						
own	1	0.879	0.802	0.759	0.727	0.661
DR's	0.514	0.524	0.585	0.572	0.593	0.642

Table 6 Example negotiation process from buyer perspective

The dealer representative's bid, his opinion of his bid, and his opinion of the bid of the buyer representative in the same round are presented in Table 7 below.

DR	round 1	2	3	4	5	accept:12
bid						
price	19777	19703	18921	19449	19282	18877
tow hedge	good	meager	meager	meager	meager	meager
airco	meager	meager	meager	meager	meager	meager
extra speakers	good	good	good	good	good	good
cd player	standard	standard	none	good	good	good
utility						
own	0.796	0.731	0.687	0.654	0.633	0.581
BR's	0.211	0.337	0.416	0.447	0.494	0.561

Table 7 Example negotiation process from dealer perspective

Note that the price attribute is monotonically increasing for the buyer' s bids, but for the dealer' s bids it does not monotonically decrease: from round 3 to 4 it increases (appearantly to compensate for a change in the CD player attribute from ' none' to ' good'). The overall utilities attributed to the own bids for both buyer and dealer are monotonic, as may be expected from the negotiation model. However, this may not be true for the utility attributed by one of the parties to the other party' s bid. Actually, from the perspective of the buyer, the dealer bid is getting a lower utility in round 4. What is perceived as a concession from one party' s perspective can provide a worse bid in the perception of the other party.

Note that the values in round 12 are such that, to the buyer representative's opinion, the utility gap has disappeared, and also the configuration gap is gone. The buyer representative, therefore, concludes after round 12 has been completed (that is: the dealer representative's reaction to his bid in round 12 has been received) that a match has been found, and asks the buyer he represents permission to close the deal instead of continuing with round 13. Given the permission of his user, the buyer representative does not call out round 13, but sends to the dealer representative an acceptance of the previous bid of the dealer representative. The dealer representative now also asks his user (the dealer) to close the deal. Given permission, the dealer representative finishes the deal with an acceptance.

Initially, the dealer asked 17,290 for the car without any accessories. For him, this would be ideal, scoring a 30% gain on the car. However, this bid is unacceptable to the buyer, who starts to negotiate. The agents quickly converge on the preferred values for tow hedge, airco, extra speakers, and cd-player, but hagle a few rounds over the price. In the final round, the buyer accepts the offer of the dealer having a car with a meager tow hedge, meager airco, good extra speakers, and good cd-player for 18,884 guilders. Based on the consumer organisations prices for such accessories, the buyer payed 2,600 for accessories, and therefore, 16,284 for the car without accessories. This means that the buyer was able to negotiate a reasonable price for himself. From the dealer' s point of view the deal is reasonable as well. He was able to sell a good CD player which he had in stock, and although he had to order the other accessories, the price still gives him a profit of 3,284 (= price – basiccost – sum of accessories = 18,884 – 13,000 – 2,600). This corresponds to a financial utility of 0.84 = profit / max financial margin * basiccost = 0.3 * 13,000). The reason that his total utility is lower (0.581) is due to the low ease utility (it takes him rather some work to do equip the car). All in all, both are satisfied.

4 Discussion

In [Gutman and Maes, 1998] a number of criteria and benefits are discussed of some different approaches to negotiation. For example, in the competitive negotiation system Kasbah three negotiation strategies are mentioned: anxious (linear increase of bids over time), cool-headed (quadratic), and frugal (exponential). In the model presented here, these strategies can be used to determine the negotiation speed. Another important issue discussed in [Gutman and Maes, 1998] is the argument for co-operative negotiation that merchants often care less about profit on any given transaction and care more about long-term profitability, which implies customer satisfaction and long-term customer relationships. That argument supports the importance of the following factors in our model for negotiation: configuration tolerance (consumer satisfaction), concession factor (profit), minimal financial margin (profit), and financial rationality (profit). Furthermore, the remark that co-operative negotiation is a win-win type of negotiation is supported by our model in that consumers and providers both have an extensive multi-attribute profile (importance factors, evaluation descriptions) that influence the outcome of the negotiation aiming to satisfy both parties.

A main difference of our work to the work described in [Benn et al., 1999] is that in our approach it is possible to specify heuristics both for the overall utilities and for separate attributes (with their values and evaluations). In their approach no overall view is made; a compensation matrix is used to compensate a concession in one attribute by other attributes. In our approach it is possible to decide about the overall concession (in terms of the overall utility) in a negotiation step, independent of specific concessions for separate attributes. Moreover, in their approach a neural (Hopfield) network is used to find the compensations for attributes by an approximation process. In contrast, our approach uses explicit knowledge to determine the attributes of a new bid, which makes it more transparent and better explainable.

In [Sierra, et al., 1998] an argumentation-based approach to negotiation is put forward. One of the issues that was left open is how the argumentation-based approach relates to utilities. This is in contrast to our approach where utilities play a main role.

Both a design description, formally specified in DESIRE, and a prototype implementation of the architecture has been constructed and tested by a group of users. Due to the various parameters making up a very detailed profile, the multi-attribute negotiation architecture for agents that is presented in this paper is more flexible with respect to user preference modelling than the existing approaches. A drawback may be, however, that to acquire such a detailed profile users may need some patience. An issue for further research is to develop automated support for this acquisition process, for example, on the basis of information acquired by monitoring the user.

The model respects the privacy of both parties (since profile information is kept local) and still is capable of adjusting to the profile of the opponent if such is desired by the user. The model allows for flexible heuristics both for the overall utilities and for the attribute evaluations. An issue for further study is how relationships between evaluations of different attributes can be exploited, for example to express that a buyer only has a high evaluation value for speakers if a CD player is present.

References

[Benn et al., 1999] W. Benn, O. Goerlitz and R. Neubert. Enabling Integrative Negotiation by Adaptive Software Agents. In: M. Klusch, O.M. Shehory, and G. Weiss (eds.). *Cooperative Information Agents III. Proceedings of the Third International Workshop on Cooperative Information Agents, CIA'99*, Lecture Notes in AI, vol. 1652, Springer Verlag, 1999, pp. 335-346.

[Brazier et al., 2000] Brazier, F.M.T., Jonker, C.M., and Treur, J. Compositional Design and Reuse of a Generic Agent Model. *Applied Artificial Intelligence Journal*, vol. 14, 2000, pp. 491-538.

[Forrester, 1997] Forrester Research Report. *Affordable Intimacy Strengthens online Stores*, 1997.

[Gutman and Maes, 1998] R. Guttman, and P. Maes. Cooperative vs. Competitive Multi-Agent Negotiation in Retail Electronic Commerce, In: *Proceedings of the Second International Workshop on Cooperative Information Agents (CIA'98)*, Paris, 1998.

[Keeney and Raifa, 1976] R. Keeney, and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. John Wiley & Sons, 1976.

[Lewicki et al., 1997] R. Lewicki, D. Saunders and J. Minton. *Essentials of Negotiation*, Irwin, 1997.

[Rosenschein and Zlotkin, 1994] J. Rosenschein, and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, 1994.

[Sierra et al., 1998] C. Sierra, N.R. Jennings, P. Noriega, and S. Parsons. A Framework for Argumentation-based Negotiation. In: M.P. Singh, A. Rao, and M.J. Wooldridge (eds.). *Intelligent Agents IV. Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages, ATAL'97*. Lecture Notes in AI, vol. 1365, Springer Verlag, pp. 177-192.

A Multiagent System for Helping Urban Traffic Management

Luis A. García and **F. Toledo**
Universitat Jaume I
12071 Castellón, Spain
{garcial, toledo}@icc.uji.es

Abstract

This paper describes the MASHGREEN DM prototype following three goals: 1) The identification and implementation of the tasks related to urban traffic control that use deep reasoning mechanisms as main tools for their resolution. The chosen model for explaining urban traffic behaviour is a qualitative model, which includes among its main features their low temporal and spatial computational costs. 2) The definition of a functional architecture with soft real time constraints that integrates the developed tasks. A specilized component named agent, composed by four functional modules executes every task. These modules are the following ones: communication protocols, methods (agent specilization), data space and control. This architecture verifies that the execution performances of every agent are not compromised by the inclusion of new agents in the system, or by the interactions due to the active system agents. 3) The implementation of a prototype in a high performance computational architecture, a *Beowulf* computer system.

1 Introduction

In the last two decades there was a broad interest in the research and development of new methods and tools suited to Intelligent Traffic Systems (ITS). However, the amount of public funds devoted to this action has been continually reduced. The main reason is that a small number of enterprises dominate the market of Urban Traffic Control Systems (UTCS). Current UTCS (including both, software and hardware components) are closed systems in the sense that they belong to the enterprises that sell them. The cities which adopt these systems have their traffic operational enlargement possibilities seriously constrained by high economic cost.

Nevertheless, the great development in the telecommunications and information technologies area, is promoting a change of mentality about the development of new ITS. This change is being used by the European Union to try to open the monopolies to the competence, as much in infrastructure supply as in the provision of services.

In the USA the *Workshop in Support ADVANCED TRAFFIC MANAGEMENT SYSTEMS (ATMS)* held in Florida, May of 1999 (funded by the *US Department of Transportation* and the *Federal Highway Administration*) pointed out the current trends in the development of new ATMS:

- The development of traffic models and the corresponding copyright of the software used to programming them. This development involves the use of quality public software (mainly *Free Software Foundation* software) to programming existing and new traffic models.
- The application of models to deal with lost of significant data to help traffic decision taken. It is important to continue the research on traffic flow and data fusion from different kinds of traffic sensors.
- The development of models and algorithms for dealing with real time characteristics. It must be identified which are the real-time operative constraints related to every task in ATMS.
- The search of models to perform dynamic settings of traffic signals. It is proposed to develop advanced traffic models related to real time traffic surveillance and automatic incident detection.

In order to follow these trends, the MASHGREEN DM prototype (A MultiAgent System for HelpinG uRban traffic dEcision takEN using Distributed Memory) prototype is developed and described in this paper. The organization of this paper is as follows. Section 2 describes the AI architectures that are being applied to traffic control. An urban traffic deep reasoning mechanism is exposed as a basic tool to deal with the above mentioned trends in ATMS. In section 3, the main features of the MASHGREEN DM prototype are exposed, including the definition of an abstract architecture suited to their agents. In section 4 the implementation and the main results of the execution of this software prototype in a *Beowulf* cluster system is exposed. The paper finishes with the main results achieved.

2 AI Architectures in UTCS

Current UTCS are generally composed of two layers. A first layer uses quantitative models, data and algorithms. A second layer uses intensively the control engineer experience. Given the complexity and the big size of the domain of the urban traffic control problem, the supervision and actions of these

engineers must be continuously provided. Therefore, the two layers UTCS are insufficient for achieving an adequate control [Bell *et al.*, 1991]. This analysis has led to several authors to introduce knowledge based systems (KBS) in the UTCS architecture [Foraste and Scemama, 1986; Bell *et al.*, 1991; Ambrosino *et al.*, 1997].

The prototypes developed up to now have been influenced by two factors: the experience learned in the implementation and evaluation of previous prototypes (considering the technologies used to manage the functionality of the system as well as the knowledge representation used) and the broad availability of faster and cheaper computers (therefore, the responses can be given early and it is possible to add new capabilities to the prototype).

The first KBS applied to traffic control (like the *SAGE* system [Foraste and Scemama, 1986]) have a shallow reasoning mechanism, that is big sets of cause/effect rules implemented in a classical functional architecture of production systems. This kind of knowledge representation is very fragile. The cause/effect rules typically reflect empirical associations derived from experience, rather than a theory of how the device under diagnosis actually works [Jackson, 1990]. Therefore, several authors proposed the integration of a model of urban traffic behaviour in the prototypes, i.e. the integration of a deep reasoning mechanism for urban traffic [Moreno *et al.*, 1993; Ambrosino *et al.*, 1997; Irgens *et al.*, 1997].

The second generation KBS integrate a deep model to explain the behaviour of the system. Besides it, they usually adopt functional collaborative architectures to integrate their components. The Blackboard architecture [Carver and Lesser, 1994] was used in the *Intelligent UTCS* prototype developed under the *DRIVE I & II* European strategic programs [Ambrosino *et al.*, 1997]. Nevertheless, these prototypes were developed and implemented using conventional computers (although the underlying theory on blackboard systems was to use a shared memory multiprocessor computer). This was due because, although there were a broad availability of shared memory parallel computers, the tools needed for programming them were not evolved at the same rate. Due to the great effort to develop better shared memory programming tools, the distributed systems were getting more significance because their lower economic cost, their performances close to the shared memory parallel computers and the wide availability of public domain tools, as MPI, for programming them.

Moreover, the prototypes based on blackboard models have a strong dependency of every component with respect to the rightness and integrity of the information stored in the blackboard, as much as its great generality, what is its strength and weakness, i.e. it is very difficult to forecast the execution sequence of their components. Therefore, it is needed a new functional collaborative architecture to reduce in a significance way the result and data dependency on execution among their main components.

The multiagent architecture fulfills these needs. The *KITS* system [Irgens *et al.*, 1997] implements an architecture called *Supervised Cooperative Agents* built on features of the *IUTCS* prototype. The key point introduced in the *KITS* system is to split the urban network in different areas. Every area is composed of a set of elements. The set of elements of each area requires a common analysis and operation process, i.e. it is an environment based architecture. Each area is managed by a different agent. The prototype exposed in this paper uses a different approach. Every agent is able to see the whole urban network. A very reliable and fast deep model reasoning is used to understand the evolution of the urban traffic. Communication among agents are performed to share knowledge and control actions. The next subsections expose the deep knowledge reasoning model used and its applications on computing several urban traffic control operations.

2.1 The Deep Reasoning Model

The main idea behind its definition is to model the spatio-temporal fluctuations in traffic levels by defining a discrete qualitative space. Every point with the same density of vehicles define a qualitative region. The spatio-temporal limit that divide two adjacent regions is represented by a straight line. The vertexes of each qualitative region are events which represent changes in the dynamics of the system: the appearance or disappearance of qualitative regions, the change from increase to decrease in the size of a qualitative region and so on. In this way, it is possible to achieve entities with cognitive meaning due to the selection of events as primitive ones that represent the evolution of the system. The urban traffic network is modeled as a set of objects (streets, intersections, traffic lights, inputs, outputs, etc.) that have associated parameters. The evolution of these parameters is described by uni- or bi-directional constraints. This qualitative simulator is broadly explained in [Moreno *et al.*, 1993].

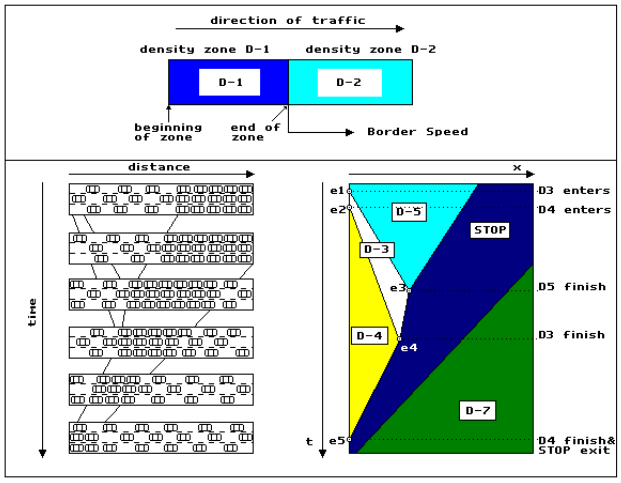


Figure 1: Temporal qualitative representation of a queue generation process due to the change from red to green in its traffic light

The evolution of the urban traffic is achieved by using the qualitative simulator and data preprocessed from sensors. They are both necessary because the current traffic status cannot be obtained just from sensors mainly for two reasons.

First, the sensors usually integrate in periods, therefore current information from sensors is not always available. Secondly, there is a limited number of sensors due to economical and operational restrictions therefore, sensors give only a small part of the information necessary to reconstruct the current traffic status. When sensor data arrives, the actual traffic status is calculated by reconstructing all the events produced during the last temporal interval without real sensor data. The time overhead of the adjustment process between the simulated and the reconstructed traffic status is small due to: (1) the time complexity of the qualitative simulator is small (quadratic on the number of intersections, segments and the length of the simulation interval in the urban network), i.e., the simulator runs much faster than real time; (2) the length of the temporal interval without real sensor data is also small (usually five minutes).

2.2 Application to UTCS Capabilities

The deep reasoning model previously exposed can be applied to several UTCS capabilities: *Preprocessing* and *Monitoring*, *Detection and Diagnosis of Problems*, *Short Term Prediction* and the *Search of Suitable Control Operations*, among others. The function for *Preprocessing* and *Monitoring* translates data from sensors into the qualitative representation described. In this way, a qualitative representation of the system independent of the sensors is obtained, which allow us to use data from several types of sensors. The current state of the system is generated by using qualitative simulation from the previous qualitative state, instead of using direct analysis of sensor data. The qualitative simulation from the previous state will give a whole image of the state of the system assuming that the previous state is correct, the prediction process is correct, and no unpredictable events, i.e. urban traffic incidents, have happened during the simulation. Sensor data is used to verify these facts when they are available.

The function for *Detection and Diagnosis of Problems* analyzes the past and present qualitative states of the system to detect problems. This task should not consider transient problems. On the other hand, a problem belonging to a zone must not be detected as the sum of the problems over the spatial components of the zone. A hierarchical classification of urban traffic problems based on the *General Representation Formalism* [Equator, 1994], an extension of the event calculus of Kowalsky with temporal granularity and continuous change, is used to accomplish this task. The matching between the traffic status reconstructed with real sensor data and the traffic status provided by the qualitative simulation is used to detect incident situations [García, 2000].

The function for *Prediction* predicts which would be the qualitative state of the system at short and medium terms. This task is done by supposing that the previous states are correct, the prediction process is correct, and have not occurred urban traffic incidents during the prediction. The low computational cost of the simulation qualitative process allows this task to be done. The aim is to detect the future problems that will arise if the current control strategy is kept. Once this overall state is computed then the system must look for problems. If there are enough detected problems then the system should search for solutions for minimizing them. This task does not need to perform a diagnosis of urban incidents process, because future detected problems are caused by bad regulation traffic settings. This behaviour is very important for an UTCS because it should have the capacity to anticipate problems. For example, one of the most common actions is to prevent a primary congestion, which has arisen as a result of the excess of traffic volume, turning into a secondary blockage. This blockage can be due to a growing queue at an intersection located afterward, with the consequent extension of the blockage over other urban network zones.

The last function exposed is the *Search of Suitable Control Operations*. If the detected problems are due to bad regulation traffic settings, then the system tries to compute other traffic settings that minimize these problems. The solution mainly consists of correcting the global effects of these bad control actions, which usually is more complicated than simply to remove the bad control actions. This is because the implementation of local solutions will produce undesirable effects, for example, that these problems will disappear from a zone but will appear in other zones of the urban network. The length reached in a traffic queue is completely determined by two conditions: the traffic status when the traffic light is green, and 2) the time allowed for red light. Therefore, it is necessary to know which are the possible red times for every intersection in the system. There are eight possible values for the red time of every input link to an intersection. The *split* of an intersection is defined as the set formed by the active red time value of every one of its input links. The set formed by the splits of all the intersections is known as a traffic plan. It is impossible to evaluate exhaustively every traffic plan, because its temporal complexity is $\Theta(8^n)$ traffic plans, where n is the number of intersections. The size of this search space is drastically reduced by using two strategies: (1) to sort the set of intersection splits in relation with the red time allowed to its main direction; and (2) to apply a branch and bound method limited to a maximum time for execution. The method consists of the qualitative simulation of the urban traffic by evaluating the profits that it can be obtained by changing the splits of the intersections in relation to their qualitative evolution [García, 2001a]. This method, with lightly modifications, can also be applied to compute other kind of control actions: values for coordination of intersections, and values for the length of the working cycles in groups of intersections [García, 2001b].

3 The MASHGREEN DM Prototype

The `MASHGREEN DM` prototype is a collaborative software architecture developed for urban traffic management tasks. Its main behaviour features are the following:

- It works with soft real-time deadlines. The execution of its tasks are interrupted at periodic time (except exceptional situations).
- It is software reusable. It can integrates tasks with different computational costs.
- It contains a heterogeneous and distributed computer implementation.

The prototype integrates software agents to perform several tasks: *monitorization* and data preprocessing, *short-medium*

term prediction of urban traffic, *bad regulation urban traffic problems identification*, *urban traffic incident detection*, continuous search of *best urban traffic plans and cycle lengths*, the *coordination* of the execution of the agents, and a friendly *interface* to the human operator. The actions allowed for the prototype to urban control are: the automatic and manual execution of the best control actions proposed by the prototype and the punctual modification of the working features of whatever controlled element in the urban network, proposed by the human operator in any moment. Therefore, the main objective of the prototype is to coordinate skills, knowledge, plans and experience among several agents to monitorize, propose and execute control actions that could improve the urban traffic status.

Four design decisions were taken in the development of the prototype: 1) organization and distribution of the data among agents, 2) integration of tasks with different computational costs, 3) implementation of a unique and common temporal reference for every agent of the prototype, and 4) the definition of an agent software architecture to be shared by every agent of the prototype.

3.1 Data Organization and Distribution

There are two kinds of data: static (does not change on execution) and dynamic (change on time). There were two options to organize the data of the prototype: 1) store all the data in the same location (e.g. using a blackboard organization) or, 2) assign different data subsets to different agents, where every agent is responsible of the integrity and correctness of the data that stores. The first option has the advantage that every agent can access to static and dynamic domain data at any time, but it also needs a complex maintenance of the data coherence. The second option overcomes this problem, but it introduces a latency in data request between agents. However, this second option also overcomes the strong centralized component that is the blackboard.

A deep study of the data dependencies among the agents shows that there are a few data which can be accessed simultaneously. Moreover, in these cases, the data access is for reading operations. Therefore, the option chosen is to replicate static data in every agent, while dynamic data are assigned to the agents that make the best use of them.

3.2 Execution Mode

Current trends in AI distributed programming pointed out the high interest in the application of distributed control methods. In these methods every agent knows when and how to perform its work. Nevertheless, the choice of a lightly centralized control has several advantages in this application domain: 1) the task of the agents must be synchronized every time the sensor data arrives to the system, therefore a coordination agent can organize these synchronizations, 2) the inner clocks of the agents must be synchronized to avoid lack of coherence between the real system and the controlled system implemented in the prototype, so a coordination agent can provide the temporal referee for every agent of the prototype, and 3) a coordination agent can be used to monitor the prototype on execution and for storing the configuration parameters of the prototype, so this feature allows us to integrate fault tolerant mechanism and to perform dynamic adding and deleting of agents. These behaviour features can be also implemented with a distributed control, but it requires a deeper knowledge in the evolution of the execution of the prototype. Therefore, the option chosen is to develop a first prototype to be improved in those undesirable behaviours as result of its later evaluation.

3.3 Common Temporal Reference

The `MASHGREEN DM` prototype implements a synchronization mechanism to maintain a common temporal clock value for every component of the prototype. This mechanism is based on the following characteristics: 1) the basic temporal unit is the second, 2) the interconnection network transmission time for a single message of whatever size is less than one second, 3) there is an agent, the coordination agent, that contains the right value for all other agents of the prototype, and 4) there is possible to apply communication primitives to perform barrier synchronization and broadcasting.

The applied method is the following:

1. The coordination agent sends at time t_i a broadcast message to request for time synchronization.
2. Every agent receives the request and sends its agreement to the coordination agent. Then, it waits until the reception of the time value message from the coordination agent.
3. The coordination agent send at time t_k the time value message when it has received the agreement message from every agent of the prototype.

Every agent that is in wait state must guarantee response to whatever message received from other agents. The returned message must notifies that the original receiver agent is executing a synchronization barrier. In this way, it is prevented deadlock situations between agents of the prototype. The method exposed has some drawbacks. The final value stored in every agent is t_k or $t_k + 1$ due to the latency in the reception of the time value message sent by the coordination agent. But this is not important due to the application domain features. Another drawback is that there is an unbalanced wasted time on the time waited for the agents until the last of them had arrived to the barrier. Therefore, to avoid these undesirable behaviour, the synchronization method is executed only at the beginning of the execution of the system and every time there were dynamic agents be added or deleted in execution time.

3.4 Agent Architecture

The software architecture of every agent is composed of four modules to deal with communications, methods, data workspace and control. The communication module implements the communication primitives (broadcasting, barrier synchronization, isolated synchronous/asynchronous sending and receiving of messages). The methods module implements two kind of methods: primary methods and secondary methods. Primary methods represent the specialization skills of the agent. Secondary methods implements administrative

work for the agent, which can be inner tasks (coherence maintenance of its data workspace) and outer tasks (knowledge coherence with the beliefs of other agents, synchronization tasks among agents and analysis of alerts). Every data belonging to the knowledge of the agent is stored in the data workspace module that is organized as a blackboard. Finally, the control module takes the decision on the action to be done at any moment depending on actual time, the messages received and the specialization skills of the agent.

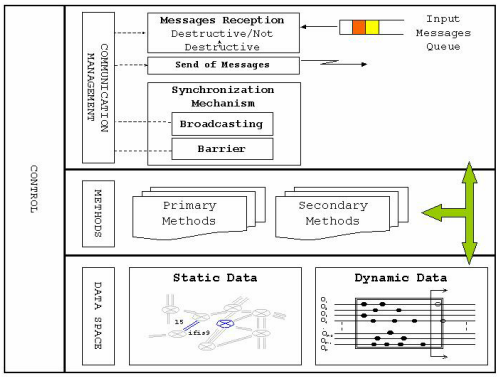


Figure 2: Module decomposition of an agent

The interaction among these modules can be described by using the following formal notation: an agent of the MASHGREEN DM prototype is a tuple of eight components: $A = (Methods, s_0, T, AgId, MsgType, K, Ags, MsgSel)$ where $Methods = PriMethods \cup SecMethods$, $s_0 \in SecMethods$ represents the initialization task of the agent, $T : Z \rightarrow Z$ is a function that returns the common virtual time of the system, $T(t_i) = t_{v_i}$, i.e. the local t_i time in an agent is the t_{v_i} virtual time common to every agent, $AgId$ is the set of possible agents identification, $MsgType$ is the finite set of kind of messages that an agent can receive or send, K is the finite set of communication primitives the agent can use, $Ags : T \rightarrow 2^{AgId}$ is the function that returns the active agents at the common temporal time t_{v_i}, and $MsgSel$ is the function that selects which message to analyze at any moment given the active set of received messages. The $MsgSel$ function defines a total order among the messages received that must fulfill two basic properties: *liveness*, i.e. every received message must be analyzed or effectively deleted, and *deadlock free*, i.e. it must not be possible to cause a deadlock situation between two or more agents. The definition of the components $Ags, T, AgId$ and K must be common to every agent belonging to the same multiagent system.

The formal definition of the *best urban traffic plan* agent is given as an example of the application of the above formal definition of the agents. In this agent the primary method is *calculate_traffic_plan*. The secondary methods are: (1) *startup_agent* $= s_0$ which prepares the agent to be included in the set of active current agents, (2) *cease_agent* which prepare the agent to be deleted of the set of current active agents, (3) *external_action* to deal with external control actions performed by the human operator, (4) *chg_mode* which set up the way that the results are communicated to other agents, and (5) *monitor* which measures and communicates to the coordination agent the main behaviour features in the execution of this agent.

Messages from other agents can be received when the agent is idle or when the agent is executing one of its methods. The $MsgSel$ function is defined in both situations. There is no mutual data dependency between this agent and whatever other agent, so this function is deadlock free. The liveness property is also fulfilled because more important messages related with the evolution and control of the traffic status are analyzed earlier, and as result of this analysis, other messages are deleted from the active set of messages received.

4 Implementation Issues

The MASHGREEN DM prototype has been implemented in a *Beowuf* computer network. This kind of computers are composed by low cost hardware components, commonly personal computers. They are programming using distribution free software, commonly *Linux* and public communication libraries. The communication workload is determined only by the application on execution. A unique node of the cluster has communication with the output network from the cluster, so the nodes of the cluster only execute tasks belonging to the cluster. The *Beowulf* cluster used is composed of one server, a biprocessor personal computer at 400 MHz, and 32 working nodes, which are personal computers working at 300 MHz. There are two active interconection networks in the system, a *Fast Ethernet* one running at 100 Mbits per second, and a *High Performance* one running at 1.2 Gbits per second. The operating system running in the *Beowulf* cluster is *Linux* and the programming language used for programming the agents is *constraint logic programming* extended with *linda* distributed programming model and *tcl/tk* script language interfaces.

The *Interface* agent runs on the server of the cluster because this agent must interact with the human operator of the prototype. The rest of agents implemented run on different working nodes of the cluster. Another working node of the cluster is used to maintain the communications in the linda model. The *Fast Ethernet* network is used for performing the communications between the agents of the prototype. This network has enough capacity because the urban traffic domain features, the lower simultaneously number of messages sent, and time needed for its transmission is not a critical feature for the agents. Moreover, the *Fast Ethernet* is a standard network model cheaper than the *High Performance* network.

Every agent of the MASHGREEN DM prototype has been evaluated using isolated and collaborative off-line executions in laboratory tests. Their executions show a stable behaviour working with different input data sets with and without the interaction with the human operator (in [García, 2001a] it is analyzed the *best urban traffic plan* agent and in [García, 2001b] it is analyzed the *best cycle length* agent). The *coordination* agent stores the monitoring messages sent by the rest of agents. These messages are stored in disk for later analysis. Figure 3 shows an example of how this analysis is done.

Figure 3: Relationship between the time used to the execution of the primary method of the most relevant agents (vertical axis) with regard to the time available when data from sensors are not available, every 300 seconds (horizontal axis). The graphic shows that the behaviour of every agent is stable

5 Conclusions

Multiagent systems are a relatively new kind of AI architectures applied to UTC that has provided a small number of prototypes up to now. The *KITS* prototype is the most referenced. In *KITS* the urban network is divided in zones. Every zone is associated with an agent that deals with every feature of its area. This is an important difference with the MASHGREEN DM prototype because, in the MASHGREEN DM prototype every agent has a domain that includes the complete urban network. Therefore, the results provided by its agents have a global view of the urban network. These agents operate by using a deep model of urban traffic which has a very low temporal computational cost. Besides that, the MASHGREEN DM prototype provides methods to integrate tasks with different computational cost. Therefore, it is possible to integrate software previously developed. Every agent of the prototype has the same architecture. This architecture is divided in four modules. The first module deals with the definition and implementation of the communication mechanism common to every agent. The second module stores the data needed to obtain its operational objectives. The data distribution among the agents allows us to increment the reliability of the system in the presence of crashed agents. Moreover, this data distribution does not significantly decrease the performances of the system, but it needs the following two conditions to be hold: (1) the agents know where every information is located: and (2) the agents know how the information is represented, i.e. they share the same data structures. The third module stores the methods suitable to be executed. These methods are primary, representing the main operational objectives of the agent, and secondary, representing administrative work for the agent. The last module deals with the control of the agent, i.e. what to do in every moment. The definition of these modules is simple to allow us the easy integration and cooperation between the agents of the prototype. The complex mechanism of their acts are settled by an inner way, depending on how these four modules are implemented and their interactions.

The MASHGREEN DM prototype has been implemented in a multiprocessor distributed computer that has three main features: (1) it has an low economic cost; (2) it has a good level of computational performances; and (3) there are stable public domain tools for programming them. The behaviour of this prototype in execution is stable, i.e. the prototype works well in laboratory tests.

Acknowledgments

This work is partly supported by the Spanish CICYT project TAP1999-0590-c02-02 and Bancaixa project P1A98-11.

References

[Ambrosino *et al.*, 1997] G. Ambrosino, M. Bielli, Boero M., Mastreta M. Application of Artificial Intelligence and Expert Systems to Traffic Control. *Advanced Vehicles and Infrastructure Systems*, John Wiley & Sons, 1997.

[Bell *et al.*, 1991] M.C. Bell, G. Scemana, L.J. Ibbetson. CLAIRE: An Expert System for Congestion Management. *Advanced Telematics in Road Transport*, Proceedings of the DRIVE conference:596-614, Elsevier, 1991.

[Carver and Lesser, 1994] N. Carver, V. Lesser. The Evolution of the Blackboard Control Architectures. *Expert Systems with Applications, Special Issue of The Blackboard Paradigm and It's Applications*, 7(1), 1994.

[Equator, 1994] EQUATOR Final. *ESPRIT p. 2049*, 1994.

[Foraste and Scemama, 1986] B. Foraste, G. Scemama. Surveillance and congested traffic control in Paris by expert system. *Second IEE Conference on Road Traffic Control*, London, 1986.

[García, 2000] L. A. García, F. Toledo. Urban Traffic Incident Detection using Qualitative Data and Temporal Intervals. In *Proc. of the IC-AI*, Las Vegas, USA, June 2000.

[García, 2001a] L. A. García, F. Toledo. An Agent for Calculating the Best Urban Traffic Plan Constrained by Soft Temporal Deadlines. In *Proc. of the SOCO/ISFI-2001*, Paisley, Scotland, June 2001.

[García, 2001b] L. A. García, F. Toledo. An Agent for Providing the Optimum Cycle Length Value in Urban Traffic Areas Constrained by Soft Temporal Deadlines. In *Proc. of the IEA/AIE-2001*, Budapest, Hungary, June 2001.

[Irgens *et al.*, 1997] M. Irgens, C. Krogh, H. Traettebers. Model Based Collaboration Architectures for Traffic Control. *Advanced Vehicles and Infrastructure Systems*, John Wiley & Sons, 1997.

[Jackson, 1990] P. Jackson. *Introduction to Expert Systems*. Addison-Wesley Publishing company, 1990.

[Moreno *et al.*, 1993] S. Moreno, F. Toledo, F. Rosisch, G. Martin. Qualitative Simulation for Temporal Reasoning in Urban Traffic Control. *Qualitative Reasoning and Decision Technologies*, N. Piera Carrete and M.G. Singh Editors, CIMNE, Barcelona, 1993.

MULTI-AGENT SYSTEMS

MULTI-AGENT SYSTEMS

Bidding Languages for Combinatorial Auctions

Craig Boutilier
Department of Computer Science
University of Toronto
Toronto, ON, M5S 3H5, CANADA
cebly@cs.toronto.edu

Holger H. Hoos
Department of Computer Science
University of British Columbia
Vancouver, BC, V6T 1Z4, CANADA
hoos@cs.ubc.ca

Abstract

Combinatorial auctions provide a valuable mechanism for the allocation of goods in settings where buyer valuations exhibit complex structure with respect to substitutability and complementarity. Most algorithms are designed to work with *explicit bids* for concrete bundles of goods. However, logical bidding languages allow the expression of complex utility functions in a natural and concise way. We introduce a new, generalized language where bids are given by propositional formulae whose subformulae can be annotated with prices. This language allows bidder utilities to be formulated more naturally and concisely than existing languages. Furthermore, we outline a general algorithmic technique for winner determination for auctions that use this bidding language.

1 Introduction

Combinatorial auctions (CAs) have been proposed as a means of dealing with the allocation of goods to buyers whose preferences exhibit complex structure with respect to complementarity and substitutability [Rassenti *et al.*, 1982; Rothkopf *et al.*, 1998; Wellman *et al.*, 2001]. Instead of selling items individually, the seller allows bids on *bundles* of items, allowing bidders to deal with the entities of direct interest and avoid the risk of obtaining incomplete bundles. Given a set of combinatorial bids, the seller then decides how best to allocate individual goods to those bundles for which bids were placed, with the aim of maximizing revenue. Because bundles generally overlap, this is—conceptually—a straightforward optimization problem, equivalent to weighted set packing. As a result, *optimal winner determination* for CAs is NP-complete [Rothkopf *et al.*, 1998].

By expressing her preferences (prices) directly over bundles, a potential buyer can, in principle, very accurately reflect her utility function, regardless of its structure. In practice, however, specifying explicit bids over all relevant bundles may be difficult: many utility functions will require the specification of a number of bundle bids that is exponential in the number of goods of interest to the bidder. This is especially true for utility functions involving the complementarities and substitutability for which CAs are best-suited. In contrast, the *logical structure* of a complex utility function might allow such preferences to be expressed relatively concisely in a suitable language. Several researchers have proposed mechanisms for expressing bids logically [Sandholm, 1999; 2000; Hoos and Boutilier, 2000; Nisan, 2000].

In this paper, we describe a generalized language for expressing bids that captures the most important elements of existing bidding languages. In our *logical combination of bids and goods* model, one specifies a bid using a logical formula, but is allowed to associate prices with arbitrary subformulae. For example, suppose in an auction for shipping capacity a bidder can send her shipment using two standard containers, a and b, or one oversized container c. The shipment has an inherent value of 50, but the convenience of using an oversize container is worth 5. We can express a suitable bid for services as $\langle\langle a \wedge b, 0\rangle \vee \langle c, 5\rangle, 50\rangle$ in our language, capturing the overall value of 50 for satisfying the requirement $a \wedge b \vee c$, as well as the premium of 5 for the oversized container. We will see examples like this below where our language allows the logical structure of a utility function to be expressed *directly* within a bid. Furthermore, our language allows one to make the very important distinctions between *sharable* and *consumable* goods, unlike existing bidding languages. We show that our language affords complete expressiveness, and that for certain natural classes of utility functions, it can express bids exponentially more compactly than existing languages. In addition, we argue that it provides a natural and concise mechanism for expressing complex bids.

We also propose an algorithmic framework for solving the winner determination problem for a set of bids expressed in our generalized logical language. We formulate a stochastic search procedure that works directly with our logical bids, sidestepping the problem of converting a logical bid into a (potentially, exponentially) large number of explicit bids. Though we have yet to study its computational properties, we expect this approach to offer a significant advance over existing algorithms.

We briefly review CAs and logical bidding languages in Section 2. In Section 3 we present our generalized logical bidding language, describing its syntax and semantics, discuss various properties of this language, and illustrate its ability to handle certain types of utility functions much more naturally and concisely than existing logical languages. In Section 4, we describe a stochastic local search procedure that exploits

the structure of our logical bids to search through the space of bid allocations to solve the winner determination problem. We conclude in Section 5 with a discussion of future work.

2 Logical Languages for Schematic Bids

In this section, we briefly review CAs and prior proposals for logical bidding languages.

2.1 Combinatorial Auctions

We suppose a seller has a set of goods $G = \{g_1, \ldots, g_n\}$ to be auctioned. Potential buyers value different subsets or *bundles* of goods, $b \subseteq G$, and offer bids of the form $\langle b, p\rangle$ where p is the amount the buyer is willing to pay for bundle b. Given a collection of bids $B = \{\langle b_i, p_i\rangle\}$, the seller must find an allocation of goods to bids that maximizes revenue. We define an *allocation* to be any $L = \{\langle b_i, p_i\rangle\} \subseteq B$ such that the bundles b_i making up L are disjoint. The *value* of an allocation $v(L)$ is given by $\sum\{p_i : \langle b_i, p_i\rangle \in L\}$. An *optimal allocation* is any allocation L with maximal value (taken over the space of allocations). The *winner determination* problem is that of finding an optimal allocation given a bid set B. We sometimes consider *assignments* $A : G \rightarrow B$ of goods to bids. Assignment A induces allocation L_A whose bids are those that have been assigned all goods (i.e., $b_i \subseteq A^{-1}(\langle b_i, p_i\rangle)$).

The winner determination problem is equivalent to the weighted set packing problem [Rothkopf *et al.*, 1998] and as such is NP-complete. Algorithms for weighted set packing and related combinatorial problems can be used for winner determination. Search algorithms—both complete methods [Fujisima *et al.*, 1999; Sandholm, 1999] as well as stochastic techniques [Hoos and Boutilier, 2000]—have been proposed in the AI literature and have proven quite successful at solving medium-sized problems. Though the problem is known not to be approximable in polynomial time, the afore-mentioned stochastic approximation technique tends to find optimal solutions very quickly for problems that can be handled by the complete methods.

2.2 Logical Languages

Most work on combinatorial auctions assumes that a bid is expressed using a simple bundle of goods associated with a price for that bundle. Such a bundle naturally captures the complementarities among the goods within that bundle. However, a buyer with a complex utility function will often need to express multiple bundle bids in order to accurately reflect her utility function.

Logical bidding languages can overcome this by allowing a bidder to express complex bids in which the logical structure of the utility function is captured. There are two distinct classes of logical bidding languages in the literature: languages that allow *logical combinations of goods* as formulae, and associate a price with each such formula (we call this the $\mathcal{L}_G$ family of languages); and languages that allow *logical combinations of bundle bids* as formulae, where the subformulae (or atomic bids) themselves have prices associated with them (we call this the $\mathcal{L}_B$ family of languages). We discuss these briefly in turn in this section, but refer to the cited papers for further details. In what follows we assume a set of goods G over which bids are expressed.

An $\mathcal{L}_G$ language is one in which logical formulae are constructed from goods; that is, goods are taken as atomic propositions and are combined using logical connectives to express a bid. A price is attached to this formula expressing the amount the bidder is prepared to offer for the *satisfaction* of that formula. Such languages can be used to capture some of the logical structure of a utility function. The language $\mathcal{L}_G^{pos}$ proposed by Hoos and Boutilier [2000] is of this type, with the restriction that only positive formulae (i.e., without negation) are considered. Formally, if $g \in G$ then $g \in \mathcal{L}_G^{pos}$; and if $\alpha_1, \alpha_2 \in \mathcal{L}_G^{pos}$, then $\alpha_1 \vee \alpha_2 \in \mathcal{L}_G^{pos}$ and $\alpha_1 \wedge \alpha_2 \in \mathcal{L}_G^{pos}$. A *logical bid* $\langle \alpha, p\rangle$ is simply a formula $\alpha \in \mathcal{L}_G^{pos}$ and an associated price p. Semantically, an assignment of goods to this bid *satisfies* the bid if the corresponding logical formula is satisfied viewing the assignment as a truth assignment (i.e., those goods assigned to the bid are "true" and those not are "false"). As an example, should a bidder desire (for price p) either g_1 or h_1, *and* g_2 or h_2, *and* g_3 or h_3, *and* g_4 or h_4, she must formulate sixteen explicit bids of the form $\{g_1, g_2, g_3, g_4\}$, $\{g_1, g_2, g_3, h_4\}$, etc. In contrast, $\mathcal{L}_G^{pos}$ allows such preferences to be expressed relatively concisely using a logical bid of the form:

$$\langle (g_1 \vee h_1) \wedge (g_2 \vee h_2) \wedge (g_3 \vee h_3) \wedge (g_4 \vee h_4), p\rangle$$

Variants of this language have been proposed by Hoos and Boutilier [2000], including the use of *k-of* clauses, expressing a desire to have any k goods from a given set, and focusing on special forms such as CNF. It is important to note that a bidder generally must express a number of logical bids in $\mathcal{L}_G^{pos}$ to capture her utility function: the fact that only one price can be attached to a formula means that independent preferences are captured by independent bids (e.g., the same bidder might bid both $\langle a, 1\rangle$ and $\langle b, 2\rangle$). While perfect substitution is captured by disjunction in $\mathcal{L}_G^{pos}$, imperfect substitutes must be dealt with using multiple bids and *dummy goods* [Fujisima *et al.*, 1999]. For example, if an agent wants only one of $a \wedge b$ or $c \wedge d$, and slightly prefers $a \wedge b$, she could specify two bids,

$$\langle a \wedge b \wedge g, p\rangle \quad \text{and} \quad \langle c \wedge d \wedge g, p'\rangle$$

with $p > p'$. The insertion of dummy good g prevents both bids from being satisfied. There is an implicit assumption of free disposal in the semantics of $\mathcal{L}_G^{pos}$. If a bid $\langle a \vee b, p\rangle$ is offered, the same price is paid if the bid is assigned a alone, b alone, or both a and b. If this assumption is violated, this bid must be broken into multiple (exclusive) bids.

A different approach is taken by Sandholm [1999; 2000] and Nisan [2000], who use $\mathcal{L}_B$ languages. Intuitively, these languages take bundle bids as their atomic elements and combine these using various logical connectives. For instance, Sandholm proposed the use of $\mathcal{L}_B^{or}$, combining atomic bids using disjunction, as in $\langle\{a, b\}, 1\rangle \vee \langle\{a, c\}, 2\rangle$. Semantically, $\mathcal{L}_B$ languages are interpreted by assigning goods to the *component atomic bids* (e.g., to $\{a, b\}$ and $\{a, c\}$ in the example above), rather than to the formula as a whole (in contrast with the $\mathcal{L}_G$ model). The price paid is determined by the logical relationship of the component bids. In $\mathcal{L}_B^{or}$, for instance, the sum of the prices of satisfied atomic bids is paid.

Several interesting varieties of $\mathcal{L}_B$ languages are studied by both Sandholm and Nisan, who consider languages using OR,

XOR ($\mathcal{L}_B^{xor}$), and two-level nesting of such connectives (OR-of-XOR and XOR-of-OR). The use of bundle bids as atomic elements allows one to express complementarities; OR (as in $\mathcal{L}_B^{or}$) allows one to capture independent preferences; and XOR allows the expression of substitutability. Nisan also proposes (and favors) the language $\mathcal{L}_B^{or*}$, essentially $\mathcal{L}_B^{or}$ with dummy goods allowed within atomic bids. $\mathcal{L}_B^{or*}$ is fully expressive and is generally more compact than the other $\mathcal{L}_B$ languages for many types of utility functions. We refer to Nisan [2000] for a discussion of the relative merits of these languages. Because multiple prices occur within a single formula, a bidder can express her preferences completely using a single bid (in contrast with the $\mathcal{L}_G$ model). However, preferences involving disjunction are often expressible much more compactly within $\mathcal{L}_G$ than $\mathcal{L}_B$. For instance, the preference function above involving multiple clauses of the form $g_i \vee h_i$ requires a bid of exponential size in any $\mathcal{L}_B$ language.

3 A Generalized Language for Logical Bids

Both language families $\mathcal{L}_G$ and $\mathcal{L}_B$ have certain drawbacks. In $\mathcal{L}_G$ languages a bidder who wants to offer different prices for related logical combinations of goods is forced to specify distinct bids for those combinations. This prevents the bids from exploiting any logically common substructure. $\mathcal{L}_B$ languages are unable to exploit the logical structure of disjunctive combinations of goods that exhibit perfect substitutability. Furthermore, $\mathcal{L}_B$ languages are unable to exploit the fact that a good may be "sharable," that is, it may contribute to the satisfaction of *multiple* atomic bids within a single bidder's logical bid. This is due to the fact that each good is assigned to an atomic bid within the $\mathcal{L}_B$ semantics and cannot be shared. This can be a severe drawback. Consider an example in which a bidder desires a single reusable resource, say a machine m, and some number of consumable resources, say raw materials r_1, r_2, etc. to be processed on m. The bidder may value each of the r_i independently, but only if the machine is available on which to process these materials. In such a case, it is most natural to express preferences for the $\langle m, r_i \rangle$ pairs, allowing m to contribute to the satisfaction of *each* such bid or subformula.

In this section, we introduce the language $\mathcal{L}_{GB}$ of *generalized logical bids* that allows for the logical combination of both goods and bids within a single formula. Specifically, a positive propositional formula over goods may have prices associated with arbitrary subformulae. As such, both goods and bids can be combined in arbitrary ways. We will see that the expressive power afforded by this approach offers a number of advantages, both in terms of the naturalness and conciseness of the representation of certain classes of utility functions. It inherits the fundamental advantages of both $\mathcal{L}_G$ and $\mathcal{L}_B$ languages.

Our new language will allow us to formulate logical bids that reflect such structured utility functions directly and concisely. We first describe the syntax and semantics of our language in fairly abstract terms. We then discuss various formal and informal properties of our language, comparing it to the languages mentioned above in terms of expressiveness, naturalness, and conciseness.

3.1 Syntax

Let G denote the set of goods, forming the atomic elements of our language. The language of $\mathcal{L}_{GB}$ is defined as follows:

- $\langle g, p \rangle \in \mathcal{L}_{GB}$, for any good $g \in G$ and any non-negative price $p \in \mathbf{R}_0^+$.
- If $b_1, b_2 \in \mathcal{L}_{GB}$, then $\langle b_1 \wedge b_2, p \rangle$, $\langle b_1 \vee b_2, p \rangle$, and $\langle b_1 \oplus b_2, p \rangle$ are all in $\mathcal{L}_{GB}$ for any non-negative price p.

Bids so-defined correspond to arbitrary propositional formulae over the goods, using connectives $\wedge$ (conjunction), $\vee$ (disjunction) and $\oplus$ (*valuative XOR*, the naming of which will become clear below), where each subformula is annotated with a price. We often don't mention the price for a subformula if $p = 0$, and call such a subformula *priceless*. A sentence $b \in \mathcal{L}_{GB}$ is called a *generalized logical bid (GLB)*. Examples of GLBs include

$$\langle \langle a, 1 \rangle \wedge \langle b, 2 \rangle, 5 \rangle \quad \text{and} \quad \langle a \vee b, 2 \rangle \oplus \langle c, 3 \rangle.$$

The *formula associated with* b, denoted $\Phi(b)$, is the logical formula obtained by removing all prices from subformulae.

3.2 Semantics

The semantics of GLBs defines the price to be paid by a bidder given a particular assignment of goods to her GLB. Roughly, the underlying idea is that the *value* of a GLB b is given by summing the prices associated with all satisfied subformulae (with one exception). We first define what it means for an assignment to satisfy a (priceless) formula.

Let A be an assignment $A : G \rightarrow B$ of goods to GLBs. Let $\Phi(b)$ be the formula associated with b. We write $\sigma(\Phi(b), A) = 1$ to denote that A *satisfies* b, and $\sigma(b, A) = 0$ to denote that A does not satisfy b. The satisfaction relation σ is defined as follows:

- If $\Phi(b) = g$ for some $g \in G$ then $\sigma(\Phi(b), A) = 1$ iff $A(g) = b$.
- If $\Phi(b) = \Phi_1 \vee \Phi_2$ then $\sigma(\Phi(b), A) = \max(\sigma(\Phi_1, A), \sigma(\Phi_2, A))$
- If $\Phi(b) = \Phi_1 \oplus \Phi_2$ then $\sigma(\Phi(b), A) = \max(\sigma(\Phi_1, A), \sigma(\Phi_2, A))$
- If $\Phi(b) = \Phi_1 \wedge \Phi_2$ then $\sigma(\Phi(b), A) = \min(\sigma(\Phi_1, A), \sigma(\Phi_2, A))$

Notice that the satisfaction relation is identical for the connectives $\vee$ and $\oplus$. The difference between the connectives will become evident when we define the value of a bid.

Given a bid b and assignment A of goods to bids, we define the *value of* b *under* A, denoted $\Psi(b, A)$, recursively. If g is a good, b_1, b_2 are bids, and p is a price:

$$\Psi(\langle g, p \rangle, A) = p \cdot \sigma(g, A)$$
$$\Psi(\langle b_1 \wedge b_2, p \rangle, A) = \Psi(b_1, A) + \Psi(b_2, A) + p \cdot \sigma(\Phi(b_1) \wedge \Phi(b_2), A)$$
$$\Psi(\langle b_1 \vee b_2, p \rangle, A) = \Psi(b_1, A) + \Psi(b_2, A) + p \cdot \sigma(\Phi(b_1) \vee \Phi(b_2), A)$$
$$\Psi(\langle b_1 \oplus b_2, p \rangle, A) = \max\{\Psi(b_1, A), \Psi(b_2, A)\} + p \cdot \sigma(\Phi(b_1) \vee \Phi(b_2), A)$$

Intuitively, the value of a bid is the value of its components, together with the additional price p if certain logical conditions are met. $\langle b_1 \wedge b_2, p\rangle$ pays price p if the formulae associated with both b_1 and b_2 are both satisfied; $\langle b_1 \vee b_2, p\rangle$ and $\langle b_1 \oplus b_2, p\rangle$ both pay price p if either (or both) of b_1 or b_2 are satisfied. The semantics of $\vee$ and $\oplus$ differ in how subformula value is used. Specifically, the value of a disjunctive bid given an assignment is the sum of the values of the subformulae: in this sense, both subformulae are of value to the bidder. In contrast, a *valuative XOR* bid, or VXOR bid, allows only the maximum value of its subformulae to be paid: thus the subformulae are viewed as substitutes (see below).

Note how, under this definition, zero prices can be used to represent subformulae to which no price is attached (such as for the conjunctive bids defined in the previous section). Let '$\equiv$' denote semantic equivalence (i.e., two GLBs have exactly the same value under any assignment of goods). We summarize the intuitive semantics of $\mathcal{L}_{GB}$ (examples are provided in the next section):

- Bids $b_1 \wedge b_2$ and $b_1 \vee b_2$ are both valued as the sum of component values for b_1 and b_2 (i.e., their utilities are independent). Without prices, conjunction and disjunction, when they appear at the top-level of a GLB, are semantically equivalent. When such formulae are priced (e.g., $\langle b_1 \wedge b_2, p\rangle$), or when they appear as subformulae in a more complex GLB, however, the meaning is quite different.
- Bid $\langle b_1 \wedge b_2, p\rangle$ expresses the complementarity of b_1 and b_2 (with value p). However, it allows intrinsic value to be expressed within the subbids (see below).
- Bid $\langle b_1 \vee b_2, p\rangle$ expresses the *partial* substitutability of b_1, b_2 (with value p). However, it allows intrinsic value to be expressed within the subbids, so satisfying both may have greater value than satisfying either one alone. If b_1, b_2 are completely priceless, then disjunction represents full and perfect substitutability.
- Bid $(b_1 \oplus b_2)$ expresses the *complete* substitutability of b_1, b_2. Only one of the values of b_1 or b_2 can be paid. If the values are distinct then they are imperfect substitutes. Perfect substitutes can be captured using $\oplus$ or $\vee$ (see above).
- Bid $\langle b_1 \oplus b_2, p\rangle$ is much like $(b_1 \oplus b_2)$, but with price p paid if either *or both* b_1, b_2 are satisfied. No "penalty" is paid if both are satisfied, so there is an implicit assumption of free disposal. A variation we do not pursue here would pay p iff *one* of the subbids held. In what follows, we assume $\oplus$ formulae have no prices, since $\langle\langle \alpha_1, p_1\rangle \oplus \langle \alpha_2, p_2\rangle, p\rangle \equiv (\langle \alpha_1, p_1 + p\rangle \oplus \langle \alpha_2, p_2 + p\rangle)$.

3.3 Properties and Examples

We begin by illustrating the key features of the generalized language $\mathcal{L}_{GB}$ with several examples. We then describe some of the formal properties of our language.

The ability to associate prices with subformulae gives $\mathcal{L}_{GB}$ the ability to express certain complex preferences much more concisely and naturally than existing languages in either the $\mathcal{L}_G$ of $\mathcal{L}_B$ families. And complex preferences often exhibit considerable structure, as studied in multiattribute utility theory [Keeney and Raiffa, 1976], that can be exploited by $\mathcal{L}_{GB}$. To illustrate, consider the bid

$$\langle\langle a, 1\rangle \wedge \langle b, 1\rangle \wedge \langle c, 3\rangle \wedge \langle d, 5\rangle, 50\rangle$$

Intuitively, this might reflect that a, b, c, and d are complementary goods with joint value 50, and that the individual goods have some intrinsic (e.g., salvage) value over and above that of their role within the group. The use of subformula prices allows the direct expression of the natural decomposition of the underlying utility function. This bid can be expressed reasonably concisely in $\mathcal{L}_B$ (e.g., OR*) and, hence, $\mathcal{L}_G$: the disjunction of five atomic bids—$\langle a, 1\rangle$, $\langle b, 1\rangle$, $\langle c, 3\rangle$, $\langle d, 4\rangle$, $\langle abcd, 60\rangle$—would suffice. However, this set of bids disguises the true structure of the utility function. Moreover, if the atoms were replaced by disjunctive formulae, the required size of the $\mathcal{L}_B$ or $\mathcal{L}_G$ formulae would blow up, since we would need to distribute the disjunction across each of the conjuncts to form suitable atomic bids.

A related bid is $\langle\langle a, 1\rangle \vee \langle b, 1\rangle \vee \langle 3, c\rangle \vee \langle d, 5\rangle, 50\rangle$. Here the individual goods are substitutes: they provide a basic functionality of value 50, but perhaps do so with differing quality (or each has different intrinsic value) reflected in the "bonus" associated with each good. Once again the use of subformula prices allows one to express this preference naturally. The most natural way to express this bid in $\mathcal{L}_B^{or*}$ would be as the disjunction of all 15 combinations of the four goods (in $\mathcal{L}_G$ this would require 15 bids instead of 15 disjuncts). In general, this would require an exponential blowup of the bid. It turns out one can express this bid more concisely as the disjunction of the following 8 bids (where g is a dummy good):[1] $\langle ag, 51\rangle$, $\langle bg, 51\rangle$, $\langle cg, 53\rangle$, $\langle dg, 55\rangle$, $\langle a, 1\rangle$, $\langle b, 1\rangle$, $\langle c, 3\rangle$, $\langle d, 5\rangle$. The dummy good is needed to ensure that 50 is not paid more than once. While the blowup is only linear, any natural structure in the utility function is buried. Furthermore, this conversion only applies when the disjuncts are goods; if they are arbitrary GLBs, then the $\mathcal{L}_B$ (or $\mathcal{L}_G$) expression will blow up.

An important feature of the semantics of $\mathcal{L}_{GB}$ is that goods are assigned to logical bids (as in $\mathcal{L}_G$) as opposed to component subformulae (as in $\mathcal{L}_B$). This means that a good assigned to a logical bid makes all occurrences of that good "true". This allows the natural distinction between "sharable" resources that complement multiple goods, and "consumable" resources whose utility can only be "counted" once. Consider a scenario in which we have a number of goods $\{r_1, \cdots, r_k\}$ whose utilities/prices p_i are conditionally *dependent* on the presence of another good m but are (conditionally) additive *independent* of each other. For instance, think of the r_i as raw materials, and of m as a machine used for processing those raw materials. This situation can be captured using a single GLB of the form:

$$\langle m \wedge r_1, p_1\rangle \vee \langle m \wedge r_2, p_2\rangle \vee \cdots \vee \langle m \wedge r_k, p_k\rangle$$

To express the same utility function using any $\mathcal{L}_B$ language would require a number of bids exponential in k (essentially requiring the enumeration of all subsets of consumable

[1] Here and in some of the following examples, in order to enhance readability, we use the notation $g_1 g_2 \ldots g_k$ instead of $\{g_1, g_2, \ldots, g_k\}$ for bundles of goods.

goods). For example, with one sharable m and four consumables r_1, r_2, r_3, r_4 (worth 1, 2, 3, and 4, respectively), we'd need the following bid (in $\mathcal{L}_B^{or}$ or $\mathcal{L}_B^{or^*}$):

$$\langle mr_1, 1\rangle \vee \langle mr_2, 2\rangle \vee \langle mr_3, 3\rangle \vee \langle mr_4, 4\rangle$$
$$\vee \langle mr_1r_2, 3\rangle \vee \langle mr_1r_3, 4\rangle \vee \langle mr_1r_4, 5\rangle \vee \langle mr_2r_3, 5\rangle$$
$$\vee \langle mr_2r_4, 6\rangle \vee \langle mr_3r_4, 7\rangle \vee \langle mr_1r_2r_3, 6\rangle \vee \langle mr_1r_2r_4, 7\rangle$$
$$\vee \langle mr_1r_3r_4, 8\rangle \vee \langle mr_2r_3r_4, 9\rangle \vee \langle mr_1r_2r_3r_4, 10\rangle$$

Again we see that $\mathcal{L}_{GB}$ allows the natural and concise expression of certain types of utility functions.

We observe that goods that complement multiple goods in a *nonsharable* fashion can be captured either using $\oplus$ or by using multiple GLBs. For instance, in the example above, if the machine m is to be treated as consumable, replacing the disjunction with VXOR:

$$\langle m \wedge r_1, p_1\rangle \oplus \langle m \wedge r_2, p_2\rangle \oplus \cdots \oplus \langle m \wedge r_k, p_k\rangle$$

would ensure that the machine was not shared across the r_i (or at least no *value* was associated with more than one r_i). Similarly, we could break up each disjunct into a separate GLB (all belonging to the same bidder): since goods are assigned to only one GLB, this approach too prevents m from being shared.

Since a bidder can offer multiple GLBs, it is important to note the distinction between the appearance of a good g in *multiple* bids and its appearance it multiple subformulae within a *single* bid. In the former case, g is treated as nonsharable, since it can be assigned to only one bid. In the latter case, each occurrence of g is satisfied by the assignment of g to that bid, hence g can be viewed as being shared by each of the component subformulae containing it. This distinction arises precisely because our notion of satisfaction is defined with respect to the assignment of goods to bids rather than bidders. We note that other *purely logical* means for distinguishing sharable and nonsharable resources may be possible, rather than relying on whether a multiple good occurrences lie "above the bid level" or below it. For instance, resource-oriented logics (e.g., linear logic [Girard, 1987]) are designed primarily to deal with the issue of resource consumption and sharing. The connections to this work seem worthy of deeper exploration.[2]

When a bidder offers multiple GLBs, we must enforce substitutability constraints by using dummy goods. This is not necessary when the bid is contained within a single GLB: VXOR can be used to ensure that only a single good from some set of (perfect or imperfect) substitutable goods is valued (assuming free disposal).

It is not hard to show that the connectives in $\mathcal{L}_{GB}$ are commutative and (in a certain sense) associative.

Proposition 1 *Let $b_1, b_2, b_3 \in \mathcal{L}_{GB}$. Then*

(a) $\langle b_2 \wedge b_1, p\rangle \equiv \langle b_1 \wedge b_2, p\rangle$ (similarly for $\vee$, $\oplus$).

(b) $\langle (b_1 \wedge b_2) \wedge b_3, p\rangle \equiv \langle b_1 \wedge (b_2 \wedge b_3), p\rangle$; note that the inner conjunctions have no price associated with them (similarly for $\vee$, $\oplus$).

[2]We note that existing resource-oriented logics do not seem to able to handle complementarities.

This justifies the informal use of conjunction (etc.) of a set of GLBs, with a price paid for the conjunction. Certain distribution laws can be derived as well for price-free subformulae, and for priced subformulae if we allow manipulation (e.g., addition and subtraction) of prices. We conjecture that several useful normal forms for $\mathcal{L}_{GB}$ exist.

While there are preferences for which $\mathcal{L}_{GB}$ offers much more concise expression than either $\mathcal{L}_G$ or $\mathcal{L}_B$, the converse is not true. Any bid expressed in $\mathcal{L}_B^{or}$, $\mathcal{L}_B^{or^*}$, $\mathcal{L}_B^{xor}$, or $\mathcal{L}_G$ can be expressed equally concisely in $\mathcal{L}_{GB}$. $\mathcal{L}_G$ bids are simply special cases of GLBs. Similarly, $\mathcal{L}_{GB}$ can represent each disjunct in an $\mathcal{L}_B^{or}$ or $\mathcal{L}_B^{or^*}$ bid as a separate GLB for the specified bidder, resulting in a collection of smaller bids whose total size, structure, and meaning is the same. As a corollary to the results of Sandholm and Nisan, that show that $\mathcal{L}_B^{xor}$ and $\mathcal{L}_B^{or^*}$ are both fully expressive, we have:

Proposition 2 *$\mathcal{L}_{GB}$ can represent any utility function over a set of goods G.*

4 Stochastic Search for GLBs

As we have seen logical languages for bid *expression* have been considered by several authors. However, the logical structure of bids formulated in these languages has not been directly exploited computationally in winner determination. For instance, in the the computational study of $\mathcal{L}_G$ languages for winner determination undertaken in [Hoos and Boutilier, 2000], compact logical bids were converted into a (large) set of explicit bids and the behavior of winner determination was examined. Despite the conversion, the stochastic local search algorithm, Casanova, proposed in that study proved to work extremely well. In this section, we formulate a stochastic search procedure for the winner determination problem for GLBs that works directly with logical bids, sidestepping the problem of converting a logical bid into a (potentially exponentially) large number of explicit bids. Though we have yet to study its computational properties, we expect this approach to offer a significant advance over existing algorithms.

Given a set B of GLBs, our aim is to find an assignment $A : G \rightarrow B$ of goods to bids that maximizes revenue; that is, whose value $V(A) = \sum_{b \in B} \Psi(b, A^{-1}(b))$ is maximal. In the spirit of the Casanova algorithm for standard combinatorial auctions, and motivated by the success of stochastic local search (SLS) techniques for a broad range of hard combinatorial problems, we devise an SLS procedure that operates directly on the space $\mathcal{A}$ of assignments and uses the objective function $V(A)$ to guide the search. In this search space, intuitively, we want to consider two assignments $A, A' \in \mathcal{A}$ to be neighbors if we can construct A from A' by shifting certain goods from some bids to others. For example, one could imagine defining the neighborhood relation as follows: a neighbor of A is reached by moving exactly one good from its assigned bid b in A to a new bid b'. While this would render each $A \in \mathcal{A}$ reachable from any other assignment, selecting good moves based on the objective function $V(A)$ would be difficult, as many single-good moves are unlikely to cause a change in $V(A)$, leading to large plateaus in the searchscape. Alternatively, one could follow the CASLS approach [Hoos and Boutilier, 2000], and consider assignments A and A' to

be neighbors if A' can be reached by selecting an unsatisfied bid b in A and shifting goods from some other bid to b so that it becomes (maximally) satisfied. We feel, however, that this approach would not adequately reflect the fact that GLBs have *degrees of satisfaction* (i.e., values). Furthermore, revenue maximizing assignments need not necessarily *maximally* satisfy any bid; thus this neighborhood relation would not necessarily allow one to reach optimal solutions from arbitrary points in search space.

The neighborhood relation we propose can be seen as a compromise between these two extremes; it is based on the observation that the existence of priced subformulae in GLBs provides the means to improve the value of a bid in natural increments by moving goods from bid to bid in *price-improving bundles*. The value of a GLB under an assignment A is determined precisely by the *priced subformulae* that are satisfied by A. We then define assignment A' to be a neighbor of A, if it can be reached from A by selecting an unsatisfied priced subformula in some bid b and by moving just enough goods to that bid to satisfy this subformula.

To complete the definition of an SLS procedure, we need to specify methods for selecting an initial assignment and for choosing a neighboring assignment at each search step. Analogous to the CASLS scheme, we propose to start the search at an empty assignment where all goods are unassigned and all the bids are fully unsatisfied (i.e., their value is zero).

To formally define the method for selecting neighbors, we use the notion of a *logical bid tree*: Each GLB b can be represented as a logical bid tree whose subtrees correspond to the priced subformulae of b; this tree is simply the parse tree for $\Phi(b)$ with prices attached to each node. A subformula without a (top-level) price attached is semantically equivalent to the same subformula with price zero; we call such subformulae and the corresponding nodes *priceless*, while all other subformulae (and nodes) are called *priced*. Since each subtree of a logical bid tree is itself a GLB, notions of value under a given assignment and maximal value under any assignment are well-defined for subtrees; the *(maximal) value of a node* is the (maximal) value of the subtree rooted at that node.

Our method for selecting a neighbor of the current assignment in each search step proceeds in two stages: First, choose from some partially unsatisfied bid b an unsatisfied price node n whose ancestors do not have maximal value.[3] Then, select a set G' of goods that, when assigned to bid b, will satisfy node n. Together, these two choices determine a neighboring assignment, which is reached by reassigning all goods in G' to bid b. More precisely, we restrict this second selection to minimal satisfying sets G', (i.e., to sets that contain only goods that are necessary to satisfy n). Finding such a minimal set satisfying n is rather straightforward using the procedure *Satisfy*(n), recursively computed as follows:

(a) Let n be a leaf node labeled with good g. Then return g.

(b) Let $n = \wedge(n_1, n_2, \cdots, n_k)$, and w.l.o.g. assume that $n_1, \cdots, n_j$ $(1 \leq j \leq k)$ are unsatisfied, and $n_{j+1}, \cdots, n_k$ are satisfied. Since n is not satisfied, at least one subnode must be unsatisfied. (Satisfaction information is recorded for each node.) Then return $\cup_{i \leq j}$ *Satisfy*(n_i), calling the *Satisfy*(n_i) in random order.

(c) Let $n = \vee(n_1, n_2, \cdots, n_k)$. (Note that since n is not satisfied, each subnode must be unsatisfied.) Randomly choose one n_i, $i \leq k$. Then return *Satisfy*(n_i).

(d) Let $n = \oplus(n_1, n_2, \cdots, n_k)$. (Note that the satisfaction of such a subformula is defined exactly as in the case of $\vee$. Hence, since n is not satisfied, each subnode must be unsatisfied.) Randomly choose one n_i, $i \leq k$. Then return *Satisfy*(n_i).

To obtain minimal assignments, we update the assignment of goods incrementally as we work recursively through the tree. This way, once a good is assigned in one part of the tree this fact will be reflected in the other parts of the tree. Due to the stochastic choice of subformula to satisfy within ORs and VXORs, and the random order in which subformulae beneath AND nodes are visited, *Satisfy*(n) can find any minimal assignment of goods to bid b that will satisfy node n.

The schematic stochastic local search algorithm we propose initializes the search at an empty assignment and then iteratively moves from the current assignment to a neighboring assignment by transferring a set of goods between bids as described above. After each such search step, the satisfaction information for all bids (and all subformulae within bids) is updated based on the new assignment. In practice, to deal with premature stagnation, this SLS technique will be extended with standard restart mechanisms such that the search is reinitialized from an empty assignment after a fixed number of steps have been performed since the last initialization (*fixed cutoff restart*) or whenever no improvement in revenue has been achieved for a given number of steps (*soft restart*).

Concrete instantiations of this algorithmic framework are obtained by specifying mechanisms for the various selections in each search step: the choice of a subformula to be satisfied, and the choice of a minimal assignment (as implemented by *Satisfy*(n), possibly extended by an additional selection from a number of minimal assignments that satisfy n). There is a broad range of possibly suitable and effective mechanisms for these selections; which of many strategies will work best will have to be determined based on empirical analyses. However, it seems clear that the choices should be made in a biased randomized fashion such that alternatives that lead to higher direct increases in $V(A)$ are selected with higher probability, while any possible alternative can be chosen with some small, lower-bounded probability. The former criterion is based on analogous results for standard combinatorial auctions [Hoos and Boutilier, 2000] and other well-known combinatorial problems such as propositional satisfiability, while the latter is a sufficient condition to ensure that for arbitrarily long runs, the SLS procedure will find an optimal solution with probability approaching one (i.e., it ensures probabilistic approximate completeness, see [Hoos, 1999] for details).

[3] To implement the search procedure efficiently, for each bid, we compute the maximum value $\max(n)$ for each node n in the logical bid tree (or for each subformula) prior to commencing the search. This computation is simple and requires just a single bottom-up sweep of each logical bid tree.

The general approach we propose here can also be used to obtain a systematic search algorithm capable of finding optimal solutions and proving their optimality. The overall search method could be very similar, starting with an empty assignment and selecting subformulae that are satisfied by assigning a set of goods in each step. To guarantee completeness of the algorithm, all choices would have to be done in a systematic fashion such that when using a backtracking mechanism, the full search space of a given problem instance will be explored after a finitely bounded amount of time. Notice that this is possible even for randomized choices. The practical efficiency of such an algorithm would depend on suitable heuristics for ordering the alternatives to be explored at each choice point, and on sufficiently powerful pruning or bounding techniques. This approach could be very useful for solving relatively small problem instances provably optimally. However, considering the NP-hardness of the given problem and well-known results for other hard combinatorial optimization problems, such as MAX-SAT, TSP, or standard combinatorial auctions, we believe that SLS techniques like the one outlined above will most likely show better absolute performance and anytime behavior on large and complex problem instances.

5 Concluding Remarks

We have proposed a new logical bidding language for CAs that exploits structure in utility functions, thereby facilitating the natural and concise expression of bids. By associating prices with subformulae and adequately dealing with sharable resources, $\mathcal{L}_{GB}$ can express certain bids exponentially more compactly than existing languages. We have also sketched a search procedure for solving CAs that does not require the conversion of logical bids to atomic bids. Though this procedure has not been tested empirically, we are confident that it will work well. We are currently developing an implementation suitable for extensive experimentation.

Apart from empirical work, we are also exploring extensions of $\mathcal{L}_{GB}$ to deal with *k-of* expressions and multiunit CAs. The distinction between sharable and consumable goods also deserves further exploration. Clearly an important concept for CAs, $\mathcal{L}_{GB}$'s ability to make this distinction implicitly is very desirable for the natural, concise expression of preferences. Finally, we are currently pursuing the connection to work in resource-oriented logics (e.g., linear logic [Girard, 1987]), though existing logics do not seem to able to handle complementarities.

References

[Fujisima *et al.*, 1999] Yuzo Fujisima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 548–553, Stockholm, 1999.

[Girard, 1987] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Hoos and Boutilier, 2000] Holger H. Hoos and Craig Boutilier. Solving combinatorial auctions using stochastic local search. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 22–29, Austin, TX, 2000.

[Hoos, 1999] Holger H. Hoos. *Stochastic Local Search – Methods, Models, Applications*. infix-Verlag, Sankt Augustin, Germany, 1999.

[Keeney and Raiffa, 1976] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, New York, 1976.

[Nisan, 2000] Noam Nisan. Bidding and allocations in combinatorial auctions. In *ACM Conference on Electronic Commerce (EC-2000)*, Minneapolis, MI, 2000.

[Rassenti *et al.*, 1982] S. J. Rassenti, V. L. Smith, and R. L. Bulfin. A combinatorial auction mechanism for airport time slot allocation. *Bell Journal of Economics*, 13:402–417, 1982.

[Rothkopf *et al.*, 1998] Michael H. Rothkopf, Aleksander Pekeč, and Ronald M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

[Sandholm, 1999] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 542–547, Stockholm, 1999. Extended version, Washington Univ. Report WUCS-99-01.

[Sandholm, 2000] Tuomas Sandholm. eMediator: a gext generation electronic commerce server. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 341–348, Barcelona, 2000.

[Wellman *et al.*, 2001] Michael P. Wellman, William E. Walsh, Peter R. Wurman, and Jeffrey K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 2001. To appear.

Partitioning Activities for Agents*†

Fatma Özcan and **V.S. Subrahmanian**
Department of Computer Science
University of Maryland, College Park
{*fatma,vs*}*@cs.umd.edu*

Abstract

There are now numerous agent applications that track interests of thousands of users in situations where changes occur continuously. [Shim *et al.*, 1994] suggested that such agents can be made efficient by merging commonalities in their activities. However, past algorithms cannot merge more than 10 or 20 concurrent activities. We develop techniques so that a large number of concurrent activities (typically over 1000) can be partitioned into components (groups of activities) of small size (e.g. 10 to 50) so that each component's activities can be merged using previously developed algorithms (e.g. [Shim *et al.*, 1994]). We first formalize the problem and show that finding optimal partitions is NP-hard. We then develop three algorithms - ***Greedy***, A^****-based*** and ***BAB*** (branch and bound). A^****-based*** and ***BAB*** are both guaranteed to compute optimal solutions. ***Greedy*** on the other hand uses heuristics and typically finds suboptimal solutions. We implemented all three algorithms. We experimentally show that the greedy algorithm finds partitions whose costs are at most 14% worse than that found by A^****-based*** and ***BAB*** — however, ***Greedy*** is able to handle over thousand concurrent requests *very fast* while the other two methods are much slower and able to handle only 10-20 requests. Hence, ***Greedy*** appears to be the best.

1 Introduction

The number of software agents deployed on the Internet has grown dramatically over the last few years. Major corporations have agents that continuously track events — some corporations use such agents to identify and exclude IP addresses that "scrape" data from their sites. Other corporations use agents that track some phenomenon (e.g. stock quotes) and take different actions for different clients who have registered parameters to be tracked and actions to be taken when certain conditions are satisfied. Yet other corporations have agents that dynamically adjust plans (e.g. airlines) when events occur that cause those plans to go awry (e.g. a snowstorm).

In this paper, we focus on agents that have a high volume of activity. *Our work is not going to help (very much) agents that have a relatively low amount of activity*. If the agent can leverage common aspects across its diverse activities so as to reduce its load, its performance will improve greatly. Much work has been done on merging activities to minimize load — such work spans databases [Shim *et al.*, 1994] and planning [Yang *et al.*, 1992] — however the problem of merging such activities in an optimal way is known to be NP-complete, and hence these algorithms tend to perform well when 10 to 20 activities or fewer are being merged. However, agents tracking user interests for major news organizations like CNN have a huge number of users with registered interests, and an enormous number of changes occurring continuously. The same is true in the case of stock market agents.

In this paper, we develop *partitioning* techniques. Such techniques work as follows: when a large set S of activities (e.g. 1000) are waiting to be performed, then our methods partition S. Such a partition breaks S into disjoint components $S_1, \ldots, S_n$ where $S = S_1 \cup \ldots \cup S_n$. The activities in a single component may then be merged by methods for merging activities, whether they are queries or plans using methods such as those of [Shim *et al.*, 1994; Yang *et al.*, 1992]. The cost of executing a component is just the cost of executing the activities in the component *after merging* activities. The cost of a partition is the sum of the costs of the components. The cost (after merging) of the activities in a partition may be estimated using techniques such as those in [Shim *et al.*, 1994; Yang *et al.*, 1992]. We would like to find a partition with minimal (estimated) cost.

Section 2 formalizes this problem and shows that finding a partition with minimal expected cost is NP-hard. Section 3 develops three algorithms to address this problem. The first algorithm is based on the $A^\star$ algorithm — we introduce a heuristic function and show that it is admissible [Nilsson, 1986]. Hence, this algorithm is guaranteed to find a partition with minimal cost. The second algorithm is a greedy algorithm which is not guaranteed to find a minimal cost partition. The third algorithm is a branch and bound algorithm which is also guaranteed to find an optimal cost solution. We have implemented all the algorithms. Section 4 describes the results

*Parts of this research were supported by Army Research Lab contract DAAL0197K0135, Army Research Office grant DAAD190010484 and DARPA/Rome Labs grant F306029910552

†Some proofs are omitted due to space limitations

of detailed experimentation with these algorithms. The experiments show that though the greedy algorithm finds suboptimal solutions (typically 14% worse savings are realized when compared to the optimal savings realizable), it is able to handle 1800 concurrent activities in about 16 seconds, while the other two algorithms are unable to handle a large number of activities.

2 Problem Definition

An agent may engage in some space of *activities*. For example, the space of activities associated with a database agent may be all SQL queries. The space of activities associated with a planning agent may be the set of all possible planning problems. The space of activities associated with a CNN-style news agent may be the set of all interests that could possibly be associated with users. In the sequel, we use $\mathcal{A}$ to denote some set of activities drawn from such an activity space. $\mathcal{A}$ denotes activities that the agent has to perform, but has not yet performed.

Definition 1 (Partition) *A partition $\mathcal{P}$ of a set $\mathcal{A}$ of activities is a set $\{A_1, A_2, \ldots, A_n\}$ where each A_i is a non-empty subset of $\mathcal{A}$ and*

1. $\mathcal{A} = A_1 \cup A_2 \cup \ldots \cup A_n$
2. $i \neq j \rightarrow A_i \cap A_j = \emptyset$.

Each A_i is called a **component** *of the partition $\mathcal{P}$. When the first condition is replaced by $\mathcal{A} \supseteq A_1 \cup A_2 \cup \ldots \cup A_n$, $\mathcal{P}$ is called a* subpartition.

A partition $\mathcal{P}$ of $\mathcal{A}$ splits a set of activities (to be done) into components. The activities in each component A_i can be separately merged, perhaps using methods for query merging [Shim *et al.*, 1994] or plan merging [Yang *et al.*, 1992].

Definition 2 (Cost estimation function) *A cost estimation function c takes a set of activities as input and returns a real number as output. c is required to satisfy at least the following axioms: (I) $c(\emptyset) = 0$, (II) $A_1 \subseteq A_2 \rightarrow c(A_1) \leq c(A_2)$.*

Intuitively, $c(A_i)$ denotes the estimated cost of executing the activities in A_i *after merging*. When $A_i = \{a_i\}$ is a singleton set, we will abuse notation and write $c(a_i)$ instead of $c(\{a_i\})$. We will also assume that the computation of c takes polynomial time. This is consistent with algorithms to estimate merged costs of sets of queries such as those of [Shim *et al.*, 1994] as well as task merging methods [Yang *et al.*, 1992]. In this paper, capitalized A's denote sets of activities. Lower case a's denote individual activities.

Problem 1 (Activity Partitioning Problem(APP)) *A set $\mathcal{A}$ of activities, a cost estimation function c, and a partition $\mathcal{P}$ of $\mathcal{A}$. Is it the case that there is no other partition $\mathcal{P}'$ of $\mathcal{A}$ such that $\sum_{A_i \in P'} c(A_i) < \sum_{A_i \in P} c(A_i)$?*

We will prove below that APP is NP-hard by using the **zero-one multiple knapsack problem (MKP)** [Martello and Toth, 1987; Kellerer, 1999]. MKP can be stated as: Given a set N of n items, a set M of m knapsacks, profit and weight vectors p_j and w_j $(1 \leq j \leq n)$, and the capacity vector c_i $(1 \leq i \leq m)$,

$$\begin{array}{lll} & max \sum_{i \in M} \sum_{j \in N} p_j x_{i,j} & \\ \text{subject to} & \sum_{j \in N} w_j x_{i,j} \leq c_i & \forall i \in M \\ & \sum_{i \in M} x_{i,j} \leq 1 & \forall j \in N \\ & x_{i,j} \in \{0,1\} & \forall i \in M, \forall j \in N \end{array}$$

The decision problem version of MKP takes in addition to the above parameters, an assignment of items to knapsacks. It returns true if the assignment is an optimal solution to the search problem and false otherwise.

Theorem 1 *The activity partitioning problem is NP-hard.*

Proof Sketch: We will transform the MKP decision problem to APP. Let m be the number of components in $\mathcal{P}$. Create three tables with the following database schemas: R_m(knapsackid,capacity), R_n(itemid, profit, weight) and R_p(knapsackid, itemid). Insert an entry for each item into R_n, and an entry for each knapsack into R_m. Create an entry in R_p for each item, knapsack pair present in the assignment α. Then, for each item i, we create an SQL query q_i given by:

> select * from R_n, R_m, R_p
> where R_p.itemid = i and R_n.itemid = R_p.itemid and R_m.knapsackid = R_p.knapsackid and sum(R_n.weight) < R_m.capacity

Let the cost, $c(q_i) = -\sum_{j \in q_i} p_j$.

It is easy to see that the tables and queries can be constructed in polynomial time. Therefore, when we solve this instance of APP, we will minimize the total cost of the queries. As a result, the sum of the profits will be maximized in the solution. Hence, APP is NP-hard. □

3 Activity Partitioning Algorithms

Although, we can transform the **MKP** problem into APP, we cannot immediately use the available approximation algorithms [Martello and Toth, 1987; Kellerer, 1999] for the **MKP** problem. This is because in *MKP* the profits of items are constants, that is they do not vary with different knapsacks. However, in our case, the cost of a set of activities not only depends on which component it belongs to, but also depends on which other activities are present in the component. Hence, we will describe *new* algorithms based on different heuristics to solve the query partitioning problem. We will start with an A*-based algorithm, which is guaranteed to find an optimal solution.

3.1 A*-based Algorithm

To adapt the A^* algorithm [Nilsson, 1986], we first need to define the state space, the cost function g and the heuristic function h.

Definition 3 (State) *A* state s *is any* subpartition *of $\mathcal{A}$. State s is a* goal state *if it is a* partition *of $\mathcal{A}$. Finally, the start state $s_0 = \emptyset$.*

Definition 4 (Functions $g(n)$, $h(n)$ and $f(n)$) *Suppose node n has state $s = \{A_1, A_2, \ldots, A_m\}$, and let*

$incr(a,s) = min\{min\{c(a \cup A_i) - c(A_i) \mid 1 \leq i \leq m\}, c(a)\}$. *Then,* $g(n)$ *and* $h(n)$ *are defined as follows:*

$$\begin{array}{ll} g(n) & = \sum_{i=1}^{m} c(A_i) \\ h(n) & = min\{incr(a,s) \mid a \in \mathcal{A} - (\bigcup_{i=1}^{m} A_i)\} \\ f(n) & = g(n) + h(n) \end{array}$$

where c *is the cost estimation function.*

Intuitively, $g(n)$ says the cost of node n is the sum of the costs of the individual components. $h(n)$ underestimates the cost to a goal state. This is done as follows. Consider each activity a that is in $\mathcal{A}$ but that is not in the subpartition associated with node n. Such an activity can either be placed in one of the components of node n or may be in a new component. Evaluate the cost of each of these alternatives and choose the minimal increase $incr_a$ in cost for adding a to node n. To obtain a partition, every activity that is not in the current subpartition must be added to node n eventually, so the cost of a solution must be at least greater than the current cost by $incr_a$ for all such activities a. This provides the rationale for $h(n)$.

Our A^****-based*** algorithm is given in Figure 1. The algorithm makes use of a function, **pick**$(s, \mathcal{A})$, that chooses an activity from $\mathcal{A}$ which has not been assigned to any of the components in state s.

If the while loop of the algorithm terminates, this implies that no goal state was reached and hence there is no solution. Note that if the heuristic function h used by the A^* algorithm is admissible, then the algorithm is guaranteed to find the optimum solution [Nilsson, 1986]. The heuristic h is admissible iff $h(n) \leq h^*(n)$ where $h^*(n)$ is the lowest actual cost of getting to a goal node from node n. The following result shows that our heuristic is admissible.

Theorem 2 (Admissibility of h**)** *For all nodes n,* $h(n) \leq h^*(n)$. *Hence,* A^****-based*** *algorithm finds an optimal partition of* $\mathcal{A}$.

Theorem 3 *The heuristic function* h *satisfies the monotone restriction.*

The monotone restriction requires that for all n, $h(n) \leq h(n') + (g(n') - g(n))$. Since, the h function satisfies the monotone restriction, the first goal state found by the A^****-based*** algorithm is guaranteed to be the optimal solution.

3.2 Greedy Algorithm

Our greedy algorithm uses the notion of a cluster graph to solve the activity partitioning problem. Given a set $\mathcal{A}$ of activities, a cluster graph is a weighted graph whose vertices are *disjoint sets* of activities. Given any set $\mathcal{A}$ of activities, we may associate with it, a *canonical cluster graph*.

Definition 5 (Canonical Cluster Graph ($\mathcal{CCG}$**))** *A* cluster graph *for a set* $\mathcal{A}$ *of activities is an undirected weighted graph where:*

1. $V = \{\{a_i\} \mid a_i \in \mathcal{A}\}$,
2. $E = \{(\{a_i\}, \{a_j\}) \mid a_i, a_j \in \mathcal{A}$ *and* $c(a_i) + c(a_j) - c(\{a_i, a_j\}) > 0\}$,
3. $w(\{a_i\}, \{a_j\}) = c(a_i) + c(a_j) - c(\{a_i, a_j\})$

Intuitively, when we have an edge between activities a_i, a_j then this means that there is some savings to be derived by merging the two activities together. The edge label, $w(a_i, a_j)$ denotes that saving. (Again, we abuse notation and write $e(a_i, a_j)$ instead of $e(\{a_i\}, \{a_j\})$ and $w(a_i, a_j)$ instead of $w(\{a_i\}, \{a_j\})$.)

```
A*-based(A)
/* Input: A (a set of activities) */
/* Output: P if one exists, NIL otherwise */

OPEN := ∅
forall a_i ∈ A do
   n_i.state := {{a_i}}
   compute g(n_i) and h(n_i)
   insert n_i into OPEN
sort OPEN in increasing order of f(n_i)
while (OPEN ≠ ∅) do
   n := OPEN.head
   delete n from OPEN
   s := n.state
   if s is a goal state then Return(s)
   else /* expand and generate children */
      a_j := pick(s, A)
      forall A_i ∈ s do
         /* insert a_j into A_i */
         s' := {A_1, ..., A_{i-1}, A_i ∪ {a_j}, ..., A_m}
         create a new node n'
         n'.state = s'
         compute g(n') and h(n')
         insert n' into OPEN
      /* also create a new component with a_j */
      create a new node n'
      n'.state := {A_1, ..., A_m, {a_j}}
      compute g(n') and h(n')
      insert n' into OPEN
end(while)
Return (NIL)
End-Algorithm
```

Figure 1: The A^*-based Algorithm

The greedy algorithm is shown in Figure 2. Intuitively, ***Greedy*** tries to build a partition iteratively. In each iteration, it finds an edge with highest weight (savings) in the cluster graph and deletes it. If more than one such edge exists, one is arbitrarily picked. It examines the two activities associated with that edge and checks to see if either of those activities already occur in a partition. There are four cases to check depending on whether one, both or neither activities occur in an existing partition. If both a_i, a_j occur in existing different components, then it may be possible to move one of them from one component into the other. This should be done only if it yields savings. If one of a_i, a_j is in an existing component but not the other, then the other can be placed in the same component. It is easy to see that the number of iterations of the loop of this algorithm is $O(card(\mathcal{A})^2)$. Each execution of the loop may take $O(card(\mathcal{P}))$ time in the worst case to check if a_i, a_j occur in a component. It is therefore easy to see that **Greedy** is certainly polynomial in the size of the input $\mathcal{A}$ *as long as*

the cost estimation function c runs in polynomial time.

Greedy($\mathcal{CCG}$)
/* **Input**: $\mathcal{CCG}$ (canonical cluster graph of $\mathcal{A}$) */
/* **Output**: $\mathcal{P}$ */

$\mathcal{P} := \emptyset$
$n := 0$
while ($\exists e \in \mathcal{CCG}$) **do**
  $e := (a_i, a_j)$, s.t. $w(e)$ is maximum in $\mathcal{CCG}$
  delete e from $\mathcal{CCG}$
  if ($\exists A_l \in \mathcal{P}\ s.t.\ a_i \in A_l$) **then**
    if ($\exists A_k \neq A_l \in \mathcal{P}\ s.t.\ a_j \in A_k$) **then**
      if ($c(A_l \cup \{a_j\}) + c(A_k - \{a_j\}) < c(A_l) + c(A_k)$)
        then delete a_j from A_k
          insert a_j into A_l
      if ($c(A_k \cup \{a_i\}) + c(A_l - \{a_i\}) < c(A_l) + c(A_k)$)
        then delete a_i from A_l
          insert a_i into A_k
    else /* $\nexists A_k\ s.t.\ a_j \in A_k$ */
      insert a_j into A_l
  else /* $\nexists A_l\ s.t.\ a_i \in A_l$ */
    if ($\exists A_k \in \mathcal{P}\ s.t.\ a_j \in A_k$) **then**
      insert a_i into A_k
    else
      $n := n + 1$
      $A_n := \{a_i, a_j\}$
      $\mathcal{P} := \mathcal{P} \cup \{A_n\}$
end(while)
Return ($\mathcal{P}$)
End-Algorithm

Figure 2: The Greedy Algorithm

3.3 Branch and Bound Algorithm

In this section, we describe a branch and bound algorithm, which is similar to the A^****-based*** algorithm, but uses a different search strategy.

The BAB algorithm maintains *nodes* n which having the following fields: ***state*** (as defined in Def. 3) , ***gval*** and ***uval*** defined below. ***gval*** specifies the value $g(n)$ (cf. Def. 4), while ***uval*** contains $u(n)$ which is defined below.

Definition 6 *Let node n have state $s = \{A_1, A_2, \ldots, A_m\}$, then $u(n) = \sum_{a_i \in \mathcal{A}'} c(a_i)$. where $\mathcal{A}' = \mathcal{A} - \bigcup_{i=1}^{m} A_i$, and c is the cost estimation function.*

Intuitively, $g(n) + u(n)$ *overestimates* the cost of the minimal cost solution reachable from node n.[1] ***BAB*** (Figure 3) also starts out with an empty partition and uses the same expansion strategy as the A^****-based*** algorithm. The OPEN list which is organized in ascending order of $g(n) + u(n)$ and the ***BAB*** algorithm always chooses the first node in OPEN for expansion. If the g value of this first node exceeds that of the best solution found so far, then we do not need to expand this node. If however, the g-value is less than that of the best solution found so far, then there is a chance that this node leads to a better solution and hence we expand it (unless it is a solution in which case we discard it).

[1]Contrast this with $h(n)$ presented earlier that represents an *underestimate*.

Branch&Bound($\mathcal{A}$)
/* **Input**: $\mathcal{A}(setofactivities)$)
/* **Output**: $\mathcal{P}$ */

create a new node n
n.state := $\{\}$
$n.uval := \sum_{a_i \in \mathcal{A}} c(a_i)$
$n.gval$:= 0
insert n into OPEN
$bestValue := n.gval + n.uval$
$bestPartition := n.state$
while (OPEN $\neq \emptyset$) **do**
  n := OPEN.head
  delete n from OPEN
  if ($n.gval < bestValue$) **then**
    if ($n.gval + n.uval < bestValue$) **then**
      $bestValue := n.gval + n.uval$
      $bestPartition := n$.state
    if (n is not a goal state) **then**
      /* expand and generate children */
      a_j := pick(s, $\mathcal{A}$)
      forall $A_i \in n.state$ **do**
        /* insert a_j into A_i */
        $s' := \{A_1, \ldots, A_{i-1}, A_i \cup \{a_j\}, \ldots, A_m\}$
        create a new node n'
        n'.state = s'
        compute $n'.gval$ and $n'.uval$
        insert n' into OPEN
      /* also create a new component with a_j */
      create a new node n'
      n'.state := $\{A_1, \ldots, A_m, \{a_j\}\}$
      compute $n', gval$ and $n'.uval$
      insert n' into OPEN
end(while)
Return($bestPartition$)
End-Algorithm

Figure 3: The Branch and Bound Algorithm

Theorem 4 ***BAB***($\mathcal{A}$) *finds an optimal partition of $\mathcal{A}$.*

4 Implementation/Experimental Results

We have implemented all three algorithms described in this paper in C++ on a Sun Ultra1 machine with 320 MB memory running Solaris 2.6. There is a total of 1600 lines of code for our implementation and experiments. We ran two sets of experiments — one to determine which algorithm was fastest, and how many concurrent activities could be handled, and another to determine how the algorithms performed in terms of savings generated by them.

Experiment 1: We first fixed an ***overlap probability***. Given a pair of activities from $\mathcal{A}$, the overlap probability gives the

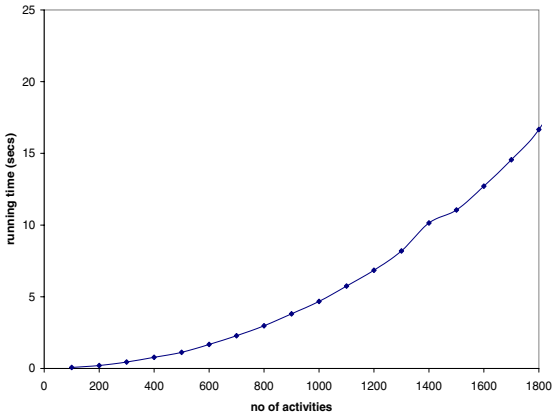


Figure 4: Scalability of the **Greedy Algorithm**

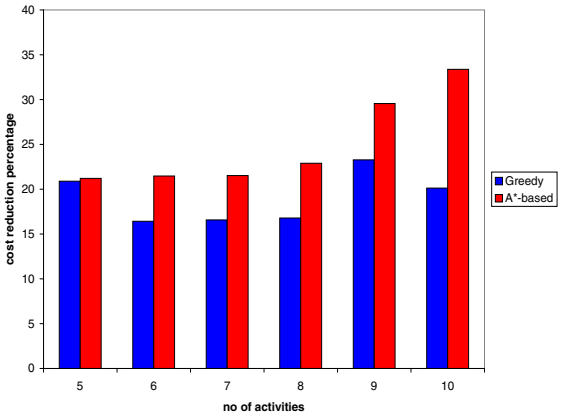


Figure 5: Cost Reduction Percentage Comparison

probability that the pair will overlap. We then fixed an ***overlap degree*** which measures the savings obtained when two activities overlap. Suppose we know activities a_1, a_2 overlap. The ***overlap degree*** in this case is $\frac{c(a_1)+c(a_2)-c(\{a_1,a_2\})}{min(c(a_1),c(a_2))}$. Intuitively, the larger the overlap degree, the greater the savings obtained by merging two activities.

Once an overlap degree and overlap probability are fixed, we automatically generated sets of activities that have that overlap degree (on the average) and that overlap probability. We then ran all three algorithms on the data. For each number of activities, we took an average over 10 sets of data. Table 1 shows the results of experiments when overlap probability varies between 0.4 and 0.6 and when the overlap degree is 0.6 (1).

No. of activities	*Greedy*	*BAB*	A^**-based*
5	2.2	81.1	68.3
6	2.75	417.1	328.5
7	2.8	1570.5	1420.7
8	3.3	6752	9941.2
9	4.7	5243.4	18281.9
10	6	10109	44782.6

Table 1: Execution Times (millisecs) (Overlap_Degree=0.6)

It turns out that both A^****-based*** and ***BAB*** ran out of memory when the number of activities exceeded about 10. The reason for this is that the OPEN list quickly grows overwhelmingly large. Hence, the above table has only 10 activities. In contrast, ***Greedy*** did not run out of space. In order to see how well ***Greedy*** could do, we continued to run experiments by increasing the number of activities. Figure 4 shows that ***Greedy*** scales well and can handle 1800 concurrent activities in about 16 seconds.

Experiment 2: As both A^****-based*** and ***BAB*** compute optimal solutions, but ***Greedy*** does not always do so, we wanted to see how much "worse" the solution produced by ***Greedy*** was compared to the optimal solution. When we have a set $\mathcal{A}$ of activities, and a partition $\mathcal{P}$ of $\mathcal{A}$, the savings realized by $\mathcal{P}$ is given by $(\Sigma_{a\in\mathcal{A}}c(a)) - (\Sigma_{A_i\in\mathcal{P}}c(A_i))$.

Figure 5 shows that in our experiments, optimal solution produced by A^****-based*** (whose savings equals that produced by ***BAB***) yielded a saving of 21–34%, while ***Greedy*** yields savings of 17–23%.

Although we could not get the optimal solution after 10 activities, we still ran the ***Greedy*** algorithm up to 1800 activities to see the quality of partitions it produced. Figure 6 shows the results. As seen from the graph, the performance of the ***Greedy*** algorithm did not degrade, and stayed about the same as we increased the number of activities in the input sets.

Both A^****-based*** and ***BAB*** are unable to perform *at all* when there are more than 10-20 concurrent activities. On larger sets of activities, they both quickly run out of memory — in contrast, ***Greedy*** scales up very effectively when many activities occur. In a few seconds, it can partition over a thousand concurrent activities, and produce a solution that is at most 40% worse than the optimal.

5 Related Work and Conclusions

Over the last few years, there has been tremendous interest in collaborative information agents. Techniques from this fertile field have found ample applications in online news sources that monitor user interests and changing news events and create personalized news reports. They have also been used extensively to make trades in financial markets for different users as stock conditions change. They have been used to dynamically bid for a variety of clients in online auctions [Schwartz and Kraus, 1997]. There are many impressive agent systems — these include *Internet Softbot* [Etzioni and Weld, 1994], *Retzina* [Decker *et al.*, 1997], Infosleuth [Bayardo et al., 1997], *SIMS* [Arens *et al.*, 1993] and many others.

There have been important efforts to optimize the performance of such agents. For instance, [Adali *et al.*, 1996; Ashish *et al.*, 1999] show how to use intelligent caching methods to improve the performance of systems that access different data sources.

In both planning and databases, researchers have noticed that when a server receives numerous requests, then it may make sense to leverage commonalities across those requests instead of serially processing the requests. This is accomplished by merging (planning requests or query requests). Important advances in this field were made by [Yang *et al.*, 1992] and [Shim *et al.*, 1994]. However, in both cases, the problem of merging plans and merging queries to minimize expected

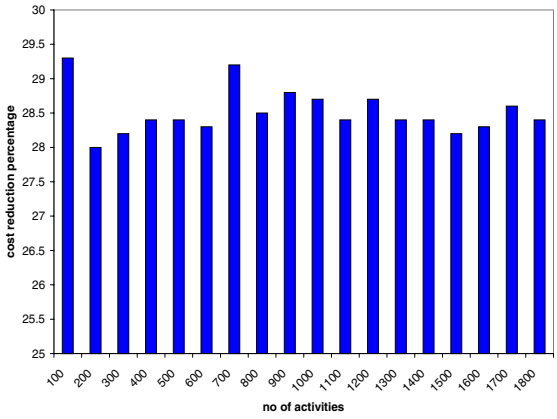


Figure 6: Cost Reduction Percentage of the **Greedy Algorithm**

cost of execution is NP-hard and hence, these methods worked well when the numbers of activities (planning or querying) were limited. [Shim *et al.*, 1994] conducted experiments with 10 concurrent queries, while [Yang *et al.*, 1992] handled up to 18 concurrent planning problems.

In this paper, we build on these important advances. Specifically, we start by assuming the existence of some way of merging k activities or fewer. Thus, if the activities are database queries, then k may be set to 10 and the [Shim *et al.*, 1994] algorithm may be used. Alternatively, if the activities are planning problems, the perhaps one could set k to 18 and use the methods of [Yang *et al.*, 1992] to merge plans. Now consider the situation where such a database or planning agent has hundreds or thousands of activities it is currently engaged in. Such a set of activities may be partitioned into components of size k (or less) so that each component's activities can be merged using an aforementioned merging algorithm such as [Yang *et al.*, 1992; Shim *et al.*, 1994]. If the sum of the costs of the individual components of the partition is minimized, then we would have an optimal way of executing such a large set of activities.

Unfortunately, this problem — which we call the activity partitioning problem (APP) in this paper — is shown to be NP-hard. As a consequence, there is no *exact* algorithm to solve this problem in polynomial time (unless P = NP which is widely believed to not be the case). We propose two exact algorithms, A^****-based*** and ***BAB***, to solve this problem. As these algorithms compute exact solutions, they are easily seen to take exponential time. As expected from earlier results of [Yang *et al.*, 1992; Shim *et al.*, 1994], these algorithms stop performing well when over 10–20 activities are being merged. To alleviate this problem, we propose a *polynomial algorithm* called ***Greedy*** which exhibits the following nice properties. First, it can (in a matter of seconds) partition a set of thousands of activities which reflects an order of magnitude improvement over A^****-based*** and ***BAB***. Second, the partitions it computes turn out to have at most 14% worse savings than those computed by A^****-based*** and ***BAB***. As a consequence, we believe that ***Greedy*** is superior to both A^****-based*** and ***BAB*** for real world problems.

References

[Adali *et al.*, 1996] S. Adali, K. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 137–148, Montreal, Canada, June 1996.

[Arens *et al.*, 1993] Y. Arens, C. Y. Chee, C.-N. Hsu, and C. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent Cooperative Information Systems*, 2(2):127–158, 1993.

[Ashish *et al.*, 1999] Naveen Ashish, Craig A. Knoblock, and Cyrus Shahabi. Selectively materializing data in mediators by analyzing user queries. In *Proceedings of the 4th IFCIS International Conference on Cooperative Information Systems, (CoopIS)*, pages 256–266, Edinburgh, Scotland, September 1999.

[Bayardo et al., 1997] R. Bayardo et al. Infosleuth: Agent-based Semantic Integration of Information in Open and Dynamic Environments. In J. Peckham, editor, *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 195–206, Tucson, Arizona, 1997.

[Decker *et al.*, 1997] K. Decker, K. Sycara, and M. Williamson. Middle Agents for the Internet. In *IJCAI*, pages 578–583, Nagoya, Japan, 1997.

[Etzioni and Weld, 1994] O. Etzioni and D. Weld. A softbot-based interface to the internet. *Communications of the ACM*, 37(7):72–76, 1994.

[Kellerer, 1999] H. Kellerer. A polynomial time approximation scheme for the multiple knapsack problem. In *Proceedings of APPROX'99*, volume 1671 of *Lecture Notes in Computer Science*, pages 51–62, Berkeley, CA, 1999.

[Martello and Toth, 1987] S. Martello and P. Toth. Algorithms for knapsack problems. In S. Martello, G. Laporte, M. Minoux, and C. Ribeiro, editors, *Annals of Discrete Mathematics 31: Surveys in Combinatorial Optimization*. North-Holland, 1987.

[Nilsson, 1986] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers, 1986.

[Schwartz and Kraus, 1997] R. Schwartz and S. Kraus. Bidding Mechanisms for Data Allocation in Multi-Agent Environments. In *Intl. Workshop on Agent Theories, Architectures, and Languages*, pages 56–70, Providence, RI, 1997.

[Shim *et al.*, 1994] K. Shim, T. K. Sellis, and D. Nau. Improvements on a heuristic algorithm for multiple-query optimization. *Data and Knowledge Engineering*, 12(2):197–222, 1994.

[Yang *et al.*, 1992] Q. Yang, D. Nau, and J. Hendler. Merging separately generated plans with restricted interactions. *Computational Intelligence*, 8(2):648–676, 1992.