Learning Behaviors in Agents Systems with Interactive Dynamic Influence Diagrams

Ross Conroy
Teesside University
Middlesbrough, UK
ross.conroy@tees.ac.uk

Yifeng Zeng
Teesside University
Middlesbrough, UK
y.zeng@tees.ac.uk

Marc Cavazza
Teesside University
Middlesbrough, UK
m.o.cavazza@tees.ac.uk

Yingke Chen
University of Georgia
Athens, GA, USA
ykchen@uga.edu

Abstract

Interactive dynamic influence diagrams (I-DIDs) are a well recognized decision model that explicitly considers how multiagent interaction affects individual decision making. To predict behavior of other agents, I-DIDs require models of the other agents to be known ahead of time and manually encoded. This becomes a barrier to I-DID applications in a human-agent interaction setting, such as development of intelligent non-player characters (NPCs) in real-time strategy (RTS) games, where models of other agents or human players are often inaccessible to domain experts. In this paper, we use automatic techniques for learning behavior of other agents from replay data in RTS games. We propose a learning algorithm with improvement over existing work by building a full profile of agent behavior. This is the first time that data-driven learning techniques are embedded into the I-DID decision making framework. We evaluate the performance of our approach on two test cases.

1 Introduction

Interactive dynamic influence diagrams (I-DIDs) [Doshi *et al.*, 2009; Zeng and Doshi, 2012] are a well recognized framework for sequential multiagent decision making under uncertainty. They are graphical decision models for individual agents in the presence of other agents who are themselves acting and observing, and whose actions affect the subject agent. They explicitly model how the other agents behave over time, based on which the subject agent's decisions are to be optimized. Importantly, I-DIDs have the advantage of a graphical representation that makes the embedded domain structure explicit by decomposing domain variables, which yields computational benefits when compared to the enumerative representation like partially observable Markov decision process (POMDP) [Smallwood and Sondik, 1973], interactive POMDP [Gmytrasiewicz and Doshi, 2005] (and so on) [Doshi *et al.*, 2009].

I-DIDs integrate two components into the decision models: one basic decision model represents the subject agent's decision making process while the other predicts behavior of other agents over time. Traditionally I-DIDs require domain experts to manually build models of other agents and solve the models in order to obtain the predicted behavior of the other agents. However, it is rather difficult to construct such models particularly in a setting of human-agent interaction where modeling humans is not easy.

In this paper, we use automatic techniques for learning behavior of other agents in I-DIDs. We will be using a real-time strategy (RTS) environment to support experiments in learning human behavior. The main advantage of using RTSs is that they provide a realistic environment in which users coexist with autonomous agents and have to make decisions in limited time in an uncertain and partially observable environment. We will use simplified situations from *StarCraft* [1] as examples for learning behaviour. The choice of *StarCraft* is motivated by the availability of replay data [Synnaeve and Bessiere, 2012], the interest it has attracted in the AI community [Ontanon *et al.*, 2013], and the applicability of learning models such as Hidden Markov models and Bayesian models [Synnaeve and Bessière, 2011], although POMDP are not directly applicable because of the size of the state space model [Ontanon *et al.*, 2013].

The learning task faces the challenge of addressing data scarcity and coverage problems because the replay data may not cover the full profile of players' behavior particularly for large planning horizons. A partial set of players' behavior needs to be expanded into a full set so that we can build a complete I-DID model. To avoid randomness of behavior expansion, we fill in the missing actions by conducting *behavioral compatibility tests* on the observed behaviors. The proposed technique is inspired by the observation that some players tend to perform identically to other players and they may execute similar actions in different game periods. Hence the missing actions will be complemented using the observed behavior. We theoretically analyze the properties of the learning technique and its influence on the planning quality. In addition, we conduct experiments in the large UAV (unmanned aerial vehicle) benchmark [Zeng and Doshi, 2012] as well as the RTS game *StarCraft*. To the best of our knowledge, it is the first time that a data-driven behavior learning technique is embedded into the I-DID model, which avoids the difficulty on manually developing models of other agents.

[1] http://eu.blizzard.com/en-gb/games/sc/

2 Related Works

Current I-DID research assumes that models of other agents can be manually built or needs to consider a large number of random candidate models where the actual models of the other agents are unknown [Zeng and Doshi, 2012]. Hence the existing I-DID research mainly focuses on reducing the solution complexity by compressing the model space of other agents and scaling up planning horizons [Doshi *et al.*, 2010; Zeng and Doshi, 2012]. The improved scalability benefits from clustering models of other agents that are behaviorally equivalent, and the behavior could be either complete or partial [Zeng *et al.*, 2011; 2012]. Recently identifying other agents' on-line behavior is used to improve I-DID solutions [Chen *et al.*, 2015], which still needs to construct candidate models of other agents. Meanwhile, Panella and Gmytrasiewicz [Panella and Gmytrasiewicz, 2015] attempt to learn models of other agents using Bayesian nonparametric methods in a small problem. Chandrasekaran *et al.* [Chandrasekaran *et al.*, 2014] intend to use reinforcement learning techniques for inferring agent behavior, which turns out to be extremely hard to reveal complete behavior.

Learning agent behavior is important in building autonomous systems, particularly with human-agent interaction [Suryadi and Gmytrasiewicz, 1999; Doshi *et al.*, 2012; Loftin *et al.*, 2014; Zilberstein, 2015], as well as developing social interactive media platforms [Salah *et al.*, 2013]. Carmel and Markovitch [Carmel and Markovitch, 1996] propose finite automata to model strategies of intelligent agents and found it difficult to learn the model. Suryadi and Gmytrasiewicz [Suryadi and Gmytrasiewicz, 1999] learn influence diagrams [Howard and Matheson, 1984] that model agents' decision making process. Instead of learning a specific model of agents, Albrecht and Ramamoorthy [Albrecht and Ramamoorthy, 2014] identify a type of agent behavior from a predefined set using a game-theoretical concept. Similarly Zhou and Yang [Zhuo and Yang, 2014] exploit action-models through transfer learning techniques to improve the agent planning quality. In parallel, learning other agents is one of the core issues in *ad-hoc* team settings - a recognized challenge in agent research [Stone *et al.*, 2010]. Barrett and Stone [Barrett and Stone, 2015] recently simplify the MDP learning to develop cooperative strategies for teammates without prior coordination.

In RTS games, Wender and Watson [Wender and Watson, 2012] use reinforcement learning to learn behavior of units in a small scale *StarCraft* combat scenario. Cho *et. al* [Cho *et al.*, 2013] take advantage of the *StarCraft* replay data that has been labelled as to the winning strategies, and improve NPC intelligence by predicting the opponent strategies.

Most of the aforementioned research still focuses on the learning of procedural models for single-agent decision making and doesn't integrate behavior learning techniques with decision models in a multiagent setting. Using a simple example inspired from *StarCraft*, we investigate the proposed learning algorithm in the I-DID framework, which is also applicable to most of general multiagent decision models that require modeling of other agents.

Figure 1: *StarCraft* Scenario (a) and unit's behavior represented by the policy tree (b).

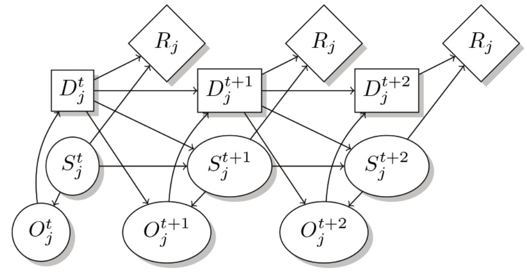


Figure 2: A DID modeling agent j in *StarCraft*.

3 Background and Initial Results

We briefly review I-DIDs through one example of a combat scenario in *StarCraft*. More formal details are found in [Doshi *et al.*, 2009; Zeng and Doshi, 2012]. After that, we provide initial results of applying I-DIDs in the game.

3.1 Interactive DIDs

Example 1 (*StarCraft* Scenario) *Fig. 1a shows a screenshot of the 1* vs *1 combat scenario. Each unit is a Terran Goliath, a unit which attacks with a cannon with a limited range of approximately half its visibility radius. Each agent can only see other agents within its visibility radius. In order for a unit to execute an attack on an enemy unit the enemy unit must be visible to the unit and be within range of the unit's weapons.*

Influence diagrams(IDs) [Howard and Matheson, 1984] use three types of nodes, namely chance node (oval), decision node (rectangle) and utility node (diamond), to represent the decision making process for a single agent in uncertain settings. When influence diagrams are unrolled over multiple time steps, they become dynamic influence diagrams (DIDs) [Tatman and Shachter, 1990] representing how the agent plans its actions sequentially.

Fig. 2 shows the DID model for controlling a unit/agent in *StarCraft*. The unit state (S^{t+1}) is whether or not the unit is under attack ($s_1 = \mathrm{UnderAttack}$, $s_2 = \mathrm{NotUnderAttack}$) which is influenced by the previous state (S^t) and action (D^t). At t the unit receives observations (O^t: $o_1 = \mathrm{See}$ and $o_2 = \mathrm{Don'tSee}$) influenced by the state (S^t) where if a unit is under attack there is a higher probability of seeing the enemy unit than when not under attack. Both the state (S^{t+1}) and reward (R) are influenced by decision (D^t: $d_1 = \mathrm{Attack}$ and $d_2 = \mathrm{Escape}$). Attacking an enemy unit gives a higher

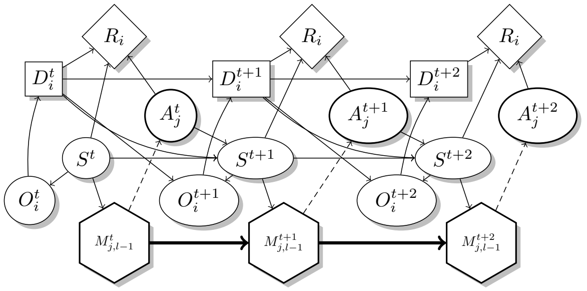


Figure 3: An I-DID modeling agent i that reasons with j's behaivor.

probability of being under attack in the next time step and lower probability for escaping. Finally the reward (R) given to the unit represents the difference in change of health for the unit and enemy unit giving a positive reward if the enemy unit looses more health than the unit and vice versa.

Extending DIDs for multiagent settings, I-DIDs introduce one new type of node, called *model node* (hexagon), that contains candidate models of other agents. Fig. 3 shows the I-DID modeling the aforementioned combat scenario in which agent i is the subject agent (NPC) in level l and agent j the other agent (player) in level $l-1$. The strategy level l allows for the nested modeling of i by the other agent j. The strategy level implies the sort of strategic reasoning between agents - what does agent i think that agent j thinks that i thinks. A level 0 agent represented by DID does not further model the other agent. The dashed arc from model node to chance node is called *policy link* that solves j's models and enters a probability distribution of j's predicted actions into the chance node. The bold arc between two model nodes is called *model update link* that specifies how j's models are updated when j acts and receives observations over time.

In the model node $M^t_{j,l-1}$ of Fig. 3, let us consider two models of agent j ($m^{t,1}_{j,l-1}$ and $m^{t,2}_{j,l-1}$) differing only in initial beliefs on the physical states. For example, the two models have the same structure of DID as shown in Fig. 2 while they have different marginal probabilities in the chance node S^t. Solving each DID model (through conventional DID techniques [Tatman and Shachter, 1990]) results in one policy tree per model. The policy tree for each model is j's optimal plan in which one action is specified for every possible observation at each time step. Fig. 1b shows the policy tree by solving the model $m^{t,1}_{j,l-1}$ of T=3 time steps.

According to the policy tree, we specify j's optimal actions in the chance node A^t_j of I-DID at t=0. After agent j executes the action at t=0 and receives one observation at t=1, it updates its beliefs in the chance node S^1 (in Fig. 2). One update of $m^{0,1}_j$ (T=3) given an observation (o_j) generates a new DID model $m^{1,1}_j$ that plans for two time steps. Consequently, four new models are generated at t=1 and placed into the model node M^1_j. Solving each of them generates optimal actions of agent j at t=1. The model update continues until only one horizon remains.

Hence solving level l I-DID involves two phases. We first need to solve models of other agents at level $l-1$ and expand the I-DID with the update of other agents' models. Then, we can transform the I-DID into a conventional DID and solve the DID to obtain the subject agent's plan.

3.2 Initial Results

We build level 0 DID (and level 1 I-DID) models for units in Fig. 1a. Initial tests show how modeling the behaviour of other agents gives a tactical advantage by predicting actions of an opponent. Different time horizons are compared to determine if there is any significant reward difference with greater time horizons. DID agents follow the generated policies in each battle. The I-DID agent has complete knowledge of all of the potential models of the DID agent and generates a policy to follow for all battles. For both battle scenarios the difference in remaining health is recorded and used to calculate an average. Table 1 shows the average health for unit i when i competes with j given: 1) both i and j are modeled by DIDs; 2) unit i is modeled by level 1 I-DID reasoning with level 0 DIDs of unit j. Results show significant advantage to the I-DID agent as a result of modeling the DID agent as part of planning.

	T=3	T=4	T=5
DID *vs* DID	19.8	13.7	35.1
I-DID *vs* DID	71.7	83.6	95.4

Table 1: Average rewards achieved by DID or I-DID controlled agent i *vs* DID controlled agent j.

Note that we do not make claims about the applicability of I-DIDs to complex, realistic *StarCraft* situations. However, DIDs are well-suited to represent the decision making process under partial observability for iterative moves, and their I-DID extension allows for the modeling of other agents. This suggests they should have the ability to learn behavior in simple configurations without suffering from the complexity of game states, and consequently that *StarCraft* can constitute an appropriate testbed for our behavior learning experiments.

4 Data-driven Behavior Learning

Given manually built models of other agents, we solve their models and integrate their solutions into the expansion of the subject agent's I-DID. However, domain knowledge is not always accessible to construct precise models of other agents. Although modeling how other agents make decisions is important in I-DID representation, the subject agent's decisions are only affected by the predicted behavior of the other agents that are solutions of the other agents' models. Hence, it will be equally effective if either the models or the behavior ascribed to the other agents are known for solving I-DIDs. With the inspiration, we learn behavior of other agents automatically, which provides an alternative to manually crafted models for solving I-DIDs. Instead of constructing and solving models to predict other agents' behavior, we learn their behavior from available data.

4.1 Behavior Representation

Solutions of a T-horizon DID are represented by a depth-T policy tree that contains a set of policy paths from the root node to the leaf. Every T-length policy path is an action-observation sequence that prescribes agent j's behavior over the entire planning horizon. Formally we define a T-length policy path below.

Definition 1 (Policy Path) *A policy path, h_j^T, is an action-observation sequence over T planning horizons: $h_j^T = \{a_j^t, o_j^{t+1}\}_{t=0}^{T-1}$, where o_j^T is null with no observations following the final action.*

Since agent j can select any action and receive every possible observation at each time step, all possible T-length policy paths are $H_j^T = A_j \times \prod_{t=1}^{T-1}(\Omega_j \times A_j)$. A depth-$T$ policy tree is the optimal plan of agent j in which the best decisions are assigned to every observation at each time step. We may impose an ordering on a policy tree by assuming some order for the observations, which guard the arcs in the tree. The total number of policy paths is up to $|\Omega_j|^{T-1}$ in a depth-T policy tree. We formally define a policy tree below.

Definition 2 (Policy Tree) *A depth-T policy tree is a set of policy paths, $\mathcal{H}_j^T$=$\bigcup h_j^T$, that is structured in a tree $\mathcal{T}$=($\mathcal{V}$,$\mathcal{E}$) where $\mathcal{V}$ is the set of vertices's (nodes) labelled with actions A and $\mathcal{E}$ is the set of ordered groups of edges labelled with observations Ω.*

Fig. 1b shows the policy tree that is the solution to one DID model in the aforementioned *StarCraft* scenario. A policy tree specifies the predicted actions of other agent j at each time step that directly impacts the subject agent i's decisions. Most of the previous I-DID research assumes the availability of domain knowledge to construct precise models of other agents, then solves the models to build the policy tree. However, this is somewhat unrealistic in RTS games if models are required to be built from the game. It is particularly difficult to model other agents that could be either other NPCs or human players in RTS games due to the complexity of their behavior descriptors.

4.2 Behavior Learning

Instead of crafting procedural models of other agents in *StarCraft*, we resort to learning other agents' behavior, generally represented by a policy tree. The possibility of learning the behavior automatically derives from the availability of replay data uploaded by a large number of game players.

Data Representation

The replay data records real-time game states, time-stamp and the status of units controlled by players. Since the replay data records the information of the entire game-play, there is no clear reset state for the units to indicate the start or end of a planning horizon. In addition, the data implicate units' behavior in many types of tasks including combat and resource gathering. It is natural that we shall focus on the learning of units' behavior for every specific task. Hence we need to partition an entire set of replay data into multiple local sets each of which supplies the corresponding data resources for learning a policy tree for a given task.

We have used a landmark approach to partition replay data, inspired from their definition in robotics [Lazanas and Latombe, 1995] and planning [Hoffmann *et al.*, 2004]. We define landmarks in *StarCraft* as a set of game features (states and values) that can be used as a point of reference to determine the player's position in the game world and the value of game-play related parameters e.g. when two enemy military units are within *Visibility Radius* they are labelled as *in battle*, this label remains until either a unit has died or both units are no longer within 1.5×*Visibility Radius*. As landmark description falls outside of the scope of this article, we shall only illustrate their role.

Constructing Policy Trees

Let $\mathcal{D} = \{\mathcal{D}_1, \cdots, \mathcal{D}_l\}$ be the entire set of replay data where $\mathcal{D}_l = \{\mathcal{D}_l^1, \cdots, \mathcal{D}_l^m\}$ aggregate the sets labelled by the same landmark l. Although the landmark decides the player's local position in the game-play, the player may have different beliefs over the game states. The local set $\mathcal{D}_l^m$ may exhibit different behavior of the player. We aim to learn a set of policy trees, $\mathcal{T} = \{\mathcal{T}_1, \cdots, \mathcal{T}_k\}$, from each $\mathcal{D}_l$.

Alg. 1 shows the development of policy trees from the data. As a policy tree is composed of the set of policy paths, we first construct the policy paths and then add them into the tree. Due to the limited interactions between NPCs and players, the replay data may not exhibit the full profile of players' behavior. Consequently, the learnt policy tree is incomplete since some branches may be missing.

Algorithm 1 Build Policy Trees

```
function LEARN POLICY TREES($\mathcal{D}$, T)
    $\mathcal{H} \leftarrow PolicyPathConstruction(\mathcal{D}, T)$
    $\mathcal{T} \leftarrow$ Build Trees From $\mathcal{H}$
    return $\mathcal{T}$
function POLICY PATH CONSTRUCTION($\mathcal{D}$, T)
    $\mathcal{H}$ set of policy paths
    for all $\mathcal{D}_l \in \mathcal{D}$ do
        New $h_j^T$
        for all $\mathcal{D}_1^m \in \mathcal{D}_1$ do
            $h_j^T \leftarrow h_j^T \cup a_j^m \, (\in \mathcal{D}_1^m[\mathcal{A}])$
            if $m \neq 1$ then
                $h_j^T \leftarrow o_j^m \, (\in \mathcal{D}_1^m[\Omega])$
        $\mathcal{H} \leftarrow \mathcal{H} \cup h_j^T$
    return $\mathcal{H}$
```

Filling in Partial Policy Trees

We can fill in the missing branches in the policy tree with random observation-action sequences, which may lead to unknown errors on predicting the agent's behavior. We observe that players may have identical post-behavior even if they act differently in the past. This is particularly true in the context of RTS games. For example, players may approach their opponents in different ways; however, they often attack them through similar mechanisms once the opponents are within the attacking distance. This inspires a heuristic for filling in the partial policy tree.

Inspired by the learning of probabilistic finite automata [Higuera, 2010], we propose the branch fill-in algorithm by testing the *behavioral compatibility* of two nodes in Alg. 2. $\mathcal{V}[\mathcal{A}^t]$ retrieve actions of time t from the policy tree while $\mathcal{V}^{\Omega}[\mathcal{A}^{t+1}]$ are actions of time t+1 given $\mathcal{V}[\mathcal{A}^t]$ with corresponding observations at $t+1$. The compatibility test (line 5) requires: (1) the nodes to have the same label(action); and (2) the difference between their successors to be bounded by a threshold value (line 20). The second condition demands the compatibility to be recursively satisfied for every pair of successor nodes (lines 10-14). Once the compatibility of the two nodes is confirmed, we can share their successors and fill in the missing branches accordingly. If multiple sets of nodes satisfy the compatibility, we select the most compatible one with the minimum difference (line 20). The time complexity of Algs. 1 and 2 is polynomial in the size of data $\mathcal{D}$.

Algorithm 2 Branch Fill-in

```
 1: function FIND MISSING NODES($\mathcal{T}$ Set of policy trees)
 2:     $\mathcal{Q}$ Set of policy trees with missing branches identified from $\mathcal{T}$
 3:     for all $\mathcal{Q}_n \in \mathcal{Q}$ do
 4:         for all $\mathcal{T}_n \in \mathcal{T}$ do
 5:             if Behavioral Compatibility($\mathcal{Q}_n$, $\mathcal{T}_n$) then
 6:                 Fill missing nodes in $\mathcal{Q}_n$ from $\mathcal{T}_n$
 7: function BEHAVIORAL COMPATIBILITY($\mathcal{V}_1$,$\mathcal{V}_2$)
 8:     if $\mathcal{V}_1[\mathcal{A}^t] \neq \mathcal{V}_2[\mathcal{A}^t]$ then return $False$
 9:     else
10:         if Test($\mathcal{V}_1, \mathcal{V}_2$) then
11:             for all $A^{t+1} \in \mathcal{V}_1$ do
12:                 $c \leftarrow$ Test($\mathcal{V}_1[A^{t+1}], \mathcal{V}_2[A^{t+1}]$)
13:                 if $c = False$ then return $False$
14:         else return $False$
15:     return $True$
16: function TEST($\mathcal{V}_1$,$\mathcal{V}_2$)
17:     $n_1 \leftarrow \mathcal{V}_1[A^t].count$, $n_2 \leftarrow \mathcal{V}_2[A^t].count$
18:     for all $\Omega \in \mathcal{V}_1$ do
19:         $f_1 \leftarrow \mathcal{V}_1^{\Omega}[A^{t+1}].count$, $f_2 \leftarrow \mathcal{V}_2^{\Omega}[A^{t+1}].count$
20:         if $|\frac{f_1}{n_1} - \frac{f_2}{n_2}| < \varepsilon$ then
21:             return false
22:     return true
```

4.3 Theoretical Analysis

The policy tree we learn from data prescribes the player's actions at every time step. Intuitively, the learnt policy tree ($\mathcal{H}_j$) will become the true behavior ($\mathcal{H}_j^*$) of the player if the amount of replay data is infinite. Let $a_j^{2,t}$ be the missing action and be filled in using action $a_j^{1,t}$ at time t in $\mathcal{H}_j$. The branch fill-in assumes that $Pr(a_j^{1,t}|\mathcal{H}_j)$ has approached the true $Pr(a_j^t|\mathcal{H}_j^*)$ after N amount of data, based on which $a_j^{2,t}$ will be complemented.

Using the *Hoeffding* inequality [Hoeffding, 1963], Proposition 1 provides the sample complexity bounding the probable rate of convergence of the fill-in actions to the true actions.

Proposition 1

$$\begin{aligned} &Pr[\textstyle\sum_{a_j^{2,t}} |Pr(a_j^{2,t}|\mathcal{H}_j) - Pr(a_j^t|\mathcal{H}_j^*)| \le \eta] \\ &\quad \ge 1 - |A_j| \cdot e^{-2NT(\frac{\eta}{|A_j|})^2} \end{aligned} \qquad (1)$$

where $Pr(a_j^{2,t}|\mathcal{H}_j)$ is the probability of actions at time t we learn from the data (computed as $\frac{f_2}{n_2}$ in line 20 of Alg. 2) and $Pr(a_j^t|\mathcal{H}_j^*)$ the true actions of the player.

Let Δ_i^T=$|V(m_{i,l}) - V^*(m_{i,l})|$ where $V(m_{i,l})$ is agent i's expected rewards by solving level l I-DID model through learning j's behavior and $V^*(m_i, l)$ is i's expected reward given j's true behavior. Following the proof in [Chen *et al.*, 2015], the reward difference is bounded below.

$$\Delta_i^T \le \rho R_i^{max}((T-1)(1 + 3(T-1)|\Omega_i||\Omega_j|) + 1) \qquad (2)$$

where $\rho = \sum_{a_j^1} |Pr(a_j^{1,t}|\mathcal{H}_j) - Pr(a_j^t|\mathcal{H}_j^*)|$ is the worst error on predicting a_j^1 in the learning.

Furthermore let $\tau = \sum_{a_j^{2,t}} |Pr(a_j^{2,t}|\mathcal{H}_j) - Pr(a_j^{1,t}|\mathcal{H}_j)|$. Since η, ρ and τ compose a triangle, we get $\rho \le \tau + \eta$ with an upper-bound τ and obtain η with probability at least $1 - |A_j| \cdot e^{-2NT(\frac{\eta}{|A_j|})^2}$.

Recall the test of line 20 in Alg. 2, we have $\tau < \mathbb{B}\varepsilon$, where $\mathbb{B}$ is the number of tests, so that we control the learning quality using the ε value in the algorithm. Note that more data will reduce the branch fill-ins therefore improving predictions of agent j's behavior, which directly impacts the agent i's plan quality in Eq. 2.

5 Experiment Results

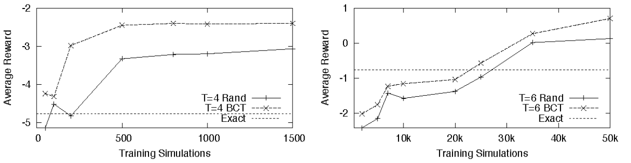
We first verify the algorithm's performance in the UAV benchmark ($|S|$=25, $|A|$=5 and $|\Omega|$=5) - the largest problem domain studied in I-POMDP/I-DID, based multiagent planning research and then demonstrate the application in *StarCraft*. We compare the policy tree learning techniques with either random fill-ins (**Rand**) or the behavioral compatibility test (**BCT**) in Alg. 2.

5.1 UAV Simulations

We simulate interactions between agents i and j, and collect different sizes of data for learning agent j's policy trees. Subsequently, we build i's I-DID given the learned j's behavior, and use the action equivalence (AE) [Zeng and Doshi, 2009] to solve the I-DID since the exact technique is not scalable to solve the complex domain. Using the generated policies, agent i plays with j that selects one model from 4 possible models of j used in the simulation.

Fig. 4a and Fig 4b shows agent i's average reward for 1000 simulations of the UAV domain for T=4 and 6 respectively. The horizontal lines (**Exact**) are agent i's average rewards when i adopts the policies generated by the I-DID, which is manually built by considering 4 possible models of agent j. The set of 4 models including j's true model are weighted uniformly in i's I-DID. The I-DID model is far more complex than what we use by learning j's behavior from the data.

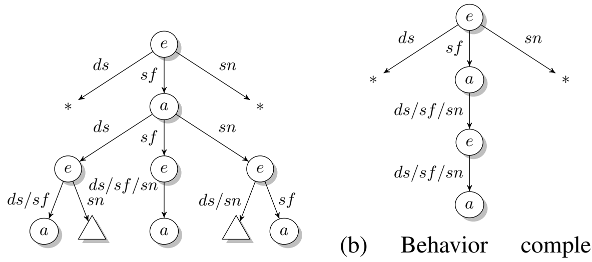
In Fig. 4, we observe that agent i achieves better performance when more data is used for learning j's behavior.



(a) Reward for I-DID agent i $T = 4$

(b) Reward for I-DID agent i $T = 6$

Figure 4: Performance of agent i by learning behavior of agent j in the UAV domain.



(a) Partial behavior

(b) Behavior complemented by BCT

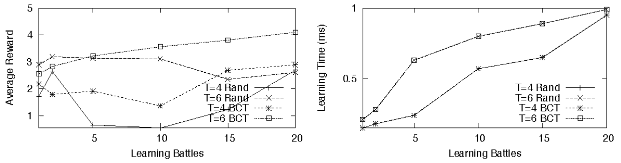
Figure 5: *StarCraft* example tree learnt from replay data with increased observations $ds = don't\ see$, $sf = see\ far$, $s = see\ near$

The learning algorithm using **BCT** outperforms the technique with **Rand** since it generates more accurate policies of agent j. The learning algorithm performs even better than the **Exact** technique because agent i can focus on the true or most probable models of agent j from learning policy trees. Negative rewards are received in Fig. 4a since it is difficult for agent i to intercept j in a short planning time (T=4). Fig. 4b shows that more data is required to learn the behavior of large planning horizons (T=6). Agent i consistently improves its rewards once more data of agent j becomes available.

5.2 Learning From *StarCraft* Replay Data

By extending the combat scenario (with more actions and observations in Example 1), we experiment with our algorithms using replay data over a number of battles. We retrieve 3 observations and 3 actions from the data, and learn the policy trees given different planning horizons. We build the learning engine using the BWAPI library [2] to interface with the games. We then develop I-DID controlled NPCs (agent i) that reason from learning behavior of other NPCs (agent j).

We learn agent j's policy trees with the planning horizons T= 4 and 6, and expand agent i's I-DID accordingly to j's policy in the low level. In Fig. 5a, we show the partial behavior of agent j learnt from the limited replay data. We use the observed actions to fill in the missing actions (denoted by triangles) under **BCT**. The complemented behavior in Fig. 5b well reflects a common battle strategy. Fig. 6a reports the average reward of agent i when it competes with agent j over 60 competitions. We observe that i receives higher reward when it learns j's behavior from more battles, where each battle records a fight/battle between 2 opposing units until one or both units dies and normally consists of 10 to 50 of j's action-observation sequences.



(a) Reward for I-DID agent i

(b) Learning Time

Figure 6: NPC (agent i) learns behavior of other NPC (agent j) from the replay data in the *StarCraft* domain.

In Fig. 6a, the results show that agent i performs better from more accurate j's behavior generated from the branch fill technique with **BCT**. Large planning horizons (like T=6) also provide some benefit to agent i. Learning time increases with more data available to the agent. Due to the inexpensive compatibility tests, the learning algorithm duration shows no visible difference in duration between random filling of missing branches compared to executing compatibility tests reflected in Fig 6b. Learning policy trees of larger planning horizons (T=6) takes more time in our experiments.

6 Conclusions

In this paper we have presented a method for learning the behaviour of a subject agent in I-DID. The approach uses automatic techniques to learn the policy tree from historical data of the agents. We have presented a solution to the problem of incomplete policy trees by way of a compatibility test to fill in missing branches with those found to be compatible. The learning technique has then been applied to both the UAV domain and the RTS game *StarCraft* to allow an NPC to predict the actions of another NPC or human player to increase its reward. Our work is an example of integrating data-driven behavior learning techniques with I-DID solutions, which removes the I-DID applications barrier of lacking models of other agents. Future work could investigate incremental algorithms for learning agents' behavior from on-line data.

[2] https://code.google.com/p/bwapi/

References

[Albrecht and Ramamoorthy, 2014] Stefano V. Albrecht and Subramanian Ramamoorthy. On convergence and optimality of best-response learning with policy types in multiagent systems. In *UAI*, pages 12–21, 2014.

[Barrett and Stone, 2015] Samuel Barrett and Peter Stone. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *AAAI*, pages 2010–2016, 2015.

[Carmel and Markovitch, 1996] David Carmel and Shaul Markovitch. Learning models of intelligent agents. In *AAAI*, pages 62–67, 1996.

[Chandrasekaran *et al.*, 2014] Muthukumaran Chandrasekaran, Prashant Doshi, Yifeng Zeng, and Yingke Chen. Team behavior in interactive dynamic influence diagrams with applications to ad hoc teams. In *AAMAS*, pages 1559–1560, 2014.

[Chen *et al.*, 2015] Yingke Chen, Yifeng Zeng, and Prashant Doshi. Iterative online planning in multiagent settings with limited model spaces and PAC guarantees. In *AAMAS*, pages 1161–1169, 2015.

[Cho *et al.*, 2013] Ho-Chul Cho, Kyung-Joong Kim, and Sung-Bae Cho. Replay-based strategy prediction and build order adaptation for starcraft ai bots. In *IEEE CIG*, pages 1–7, 2013.

[Doshi *et al.*, 2009] Prashant Doshi, Yifeng Zeng, and Qiongyu Chen. Graphical models for interactive POMDPs: Representations and solutions. *JAAMAS*, 18(3):376–416, 2009.

[Doshi *et al.*, 2010] Prashant Doshi, Muthukumaran Chandrasekaran, and Yifeng Zeng. Epsilon-subject equivalence of models for interactive dynamic influence diagrams. In *IAT*, pages 165–172, 2010.

[Doshi *et al.*, 2012] Prashant Doshi, Xia Qu, Adam Goodie, and Diana L. Young. Modeling human recursive reasoning using empirically informed interactive partially observable markov decision processes. *IEEE SMC A*, 42(6):1529–1542, 2012.

[Gmytrasiewicz and Doshi, 2005] Piotr Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multiagent settings. *JAIR*, 24:49–79, 2005.

[Higuera, 2010] Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammar*. Cambridge University Press, 2010.

[Hoeffding, 1963] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[Hoffmann *et al.*, 2004] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *JAIR*, 22:215–278, 2004.

[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, 1984.

[Lazanas and Latombe, 1995] A. Lazanas and J. C. Latombe. Landmark-based robot navigation. *Algorithmica*, 13(5):472–501, 1995.

[Loftin *et al.*, 2014] Robert Tyler Loftin, James MacGlashan, Bei Peng, Matthew E. Taylor, Michael L. Littman, Jeff Huang, and David L. Roberts. A strategy-aware technique for learning behaviors from discrete human feedback. In *AAAI*, pages 937–943, 2014.

[Ontanon *et al.*, 2013] Santiago Ontanon, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game AI research and competition in starcraft. In *IEEE CIG*, volume 5, pages 293–311, 2013.

[Panella and Gmytrasiewicz, 2015] Alessandro Panella and Piotr Gmytrasiewicz. Nonparametric bayesian learning of other agents' policies in multiagent pomdps. In *AAMAS*, pages 1875–1876, 2015.

[Salah *et al.*, 2013] Albert Ali Salah, Hayley Hung, Oya Aran, and Hatice Gunes. Creative applications of human behavior understanding. In *HBU*, pages 1–14, 2013.

[Smallwood and Sondik, 1973] Richard Smallwood and Edward Sondik. The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[Stone *et al.*, 2010] Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI*, pages 1504–1509, 2010.

[Suryadi and Gmytrasiewicz, 1999] Dicky Suryadi and Piotr J. Gmytrasiewicz. Learning models of other agents using influence diagrams. *User Modeling*, 407:223–232, 1999.

[Synnaeve and Bessière, 2011] Gabriel Synnaeve and Pierre Bessière. A Bayesian model for RTS units control applied to StarCraft. In *2011 IEEE CIG*, pages 190–196, 2011.

[Synnaeve and Bessiere, 2012] Gabriel Synnaeve and P Bessiere. A Dataset for StarCraft AI & an Example of Armies Clustering. *Artificial Intelligence in Adversarial Real-Time Games*, pages 25–30, 2012.

[Tatman and Shachter, 1990] Joseph A. Tatman and Ross D. Shachter. Dynamic programming and influence diagrams. *IEEE SMC*, 20(2):365–379, 1990.

[Wender and Watson, 2012] Stefan Wender and Ian Watson. Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft: Broodwar. In *IEEE CIG*, pages 402–408, 2012.

[Zeng and Doshi, 2009] Yifeng Zeng and Prashant Doshi. Speeding up exact solutions of interactive influence diagrams using action equivalence. In *IJCAI*, pages 1996–2001, 2009.

[Zeng and Doshi, 2012] Yifeng Zeng and Prashant Doshi. Exploiting model equivalences for solving interactive dynamic influence diagrams. *JAIR*, 43:211–255, 2012.

[Zeng *et al.*, 2011] Yifeng Zeng, Prashant Doshi, Yinghui Pan, Hua Mao, Muthukumaran Chandrasekaran, and Jian Luo. Utilizing partial policies for identifying equivalence of behavioral models. In *AAAI*, pages 1083–1088, 2011.

[Zeng *et al.*, 2012] Yifeng Zeng, Yinghui Pan, Hua Mao, and Jian Luo. Improved use of partial policies for identifying behavioral equivalences. In *AAMAS*, pages 1015–1022, 2012.

[Zhuo and Yang, 2014] Hankz Hankui Zhuo and Qiang Yang. Action-model acquisition for planning via transfer learning. *Artificial Intelligence*, 212:80–103, 2014.

[Zilberstein, 2015] Shlomo Zilberstein. Building strong semi-autonomous systems. In *AAAI*, pages 4088–4092, 2015.