Multi-Agent Intention Progression with Black-Box Agents

Michael Dann\textsuperscript{1*}, Yuan Yao\textsuperscript{2}, Brian Logan\textsuperscript{3} and John Thangarajah\textsuperscript{1}
\textsuperscript{1}RMIT University
\textsuperscript{2}Zhejiang University of Technology
\textsuperscript{3}Utrecht University

\{michael.dann, john.thangarajah\}@rmit.edu.au, yaoyuan@zjut.edu.cn, b.s.logan@uu.nl

Abstract

We propose a new approach to intention progression in multi-agent settings where other agents are effectively black boxes. That is, while their goals are known, the precise programs used to achieve these goals are not known. In our approach, agents use an abstraction of their own program called a partially-ordered goal-plan tree (pGPT) to schedule their intentions and predict the actions of other agents. We show how a pGPT can be derived from the program of a BDI agent, and present an approach based on Monte Carlo Tree Search (MCTS) for scheduling an agent’s intentions using pGPTs. We evaluate our pGPT-based approach in cooperative, selfish and adversarial multi-agent settings, and show that it out-performs MCTS-based scheduling where agents assume that other agents have the same program as themselves.

1 Introduction

A key problem for an autonomous intelligent agent with multiple goals is ‘what to do next’: which goal the agent should be trying to achieve, and which means it should use to achieve it. In the popular Belief-Desire-Intention (BDI) approach to agents [Rao and Georgeff, 1992], this problem is termed the intention progression problem (IPP) [Logan et al., 2017]. BDI agents are characterised by the concepts of beliefs, goals and plans. Beliefs represent the information an agent has about itself, the environment and about other agents. Goals are states the agent would like to bring about. Plans are recipes for achieving goals, and are composed of primitive actions that directly change the state of the environment, and subgoals which are achieved by their own plans. An intention is formed when the agent commits to achieving a (top-level) goal utilising a particular plan. A key feature of BDI agents is their ability to simultaneously pursue multiple intentions. In order to do so, at each deliberation cycle, the agent must select which of its multiple intentions it should progress (i.e., intention selection) and, if the next step within the selected intention is a subgoal, select the best plan to achieve it (i.e., plan selection). These two choices together form the intention progression problem.

A number of approaches to various aspects of the intention progression problem have been proposed in the literature, including summary-information-based (SI) [Thangarajah et al., 2003; Thangarajah and Padgham, 2011], coverage-based (CB) [Waters et al., 2014; Waters et al., 2015] and Monte-Carlo Tree Search-based (MCTS) [Yao et al., 2014; Yao and Logan, 2016; Yao et al., 2016] approaches. Much of this work has focussed on the single agent setting, where the key challenge is the interleaving of steps in plans in different intentions to avoid conflicts, i.e., when the execution of a step in one plan makes the execution of a step in another concurrently executing plan impossible. Recently, Dann et al. [2020] extended the MCTS-based approach in [Yao and Logan, 2016] to a multi-agent setting. In the multi-agent setting, how an agent progresses its intentions has implications for both the achievement of its own goals and the achievement of the goals of other agents, e.g., if the agent selects a plan that consumes a resource necessary for another agent to achieve its goal. While the ‘intention-aware’ approach in [Dann et al., 2020] was shown to out-perform non-intention-aware scheduling such as [Yao and Logan, 2016], it assumes that agents have access to the plans comprising the other agents’ programs for achieving their goals. This is reasonable in multi-agent systems where the agents are co-designed, but is less plausible for agents that are not co-designed, i.e., where each agent may have no information about the plans used by other agents.

One way of overcoming this limitation is for an agent to assume that the programs of other agents are the same as its own when progressing its intentions. However, such “ego-centric” scheduling can give rise to conflicts if the plans used by other agents achieve subgoals in a different order, or order primitive actions differently. For example, an agent $a_1$ whose plan to achieve the goal of doing the household chores contains subgoals to do the laundry, vacuum, and wash the dishes (in that order) may assume that another agent, $a_2$, tasked with doing the chores will wash the dishes last. This may result in a conflict when $a_1$ tries to use the sink to prepare lunch, if the program of $a_2$ has washing the dishes as the first subgoal.

A more reasonable assumption to make is that the other agents use a similar program, but may order some subtasks
2 Preliminaries

In this section, we introduce and define the basic elements of our approach to intention progression, including beliefs, goals, actions and plans.

Beliefs and goals. The agent’s beliefs represent the information an agent has about itself, the environment and about other agents. The agent’s belief base $B$ is a finite set of ground literals (proposition $p$ or its negation $\neg p$):

$$B = \{b_1, \ldots, b_n\}$$

$B$ is updated at each cycle to incorporate the agent’s current percepts. We assume that $B$ is consistent, i.e., there is no $p, \neg p \in B$. The agent’s top-level goals represent states of affairs that the agent wants to bring about. The agent’s goal base $G$ is a finite set of ground literals:

$$G = \{g_1, \ldots, g_m\}$$

$G$ does not need to be consistent, e.g., conflicting goals may be achieved at different times.

Actions and plans. An agent can perform a set of primitive actions in the environment, $A = \{a_1, \ldots, a_k\}$. The preconditions of an action $a_i$ are a set of literals $\phi = \text{pre}(a_i)$ that must be true before the execution of the action, and the postconditions of the action are a set of literals $\psi = \text{post}(a_i)$ that are true after the execution of the action. We assume that actions are deterministic: if the preconditions of an action hold, then the postconditions of the action hold after executing the action. An action is executable given the agent’s beliefs $B$ if $B \models \phi$. If the agent attempts to execute an action whose preconditions do not hold, the action fails (cannot be executed).

To achieve the agent’s goals, actions are organised into plans. Each goal $g$ is associated with a set of plans $\pi_1, \ldots, \pi_n$ that achieve $g$. Each plan $\pi_i$ is of the form $g : \chi \leftarrow s_1; \ldots; s_m$, where $\chi = \text{pre}(\pi_i)$ is a set of literals specifying the context condition which must be true for $\pi_i$ to begin execution, and $s_1; \ldots; s_m$ is a sequence of steps which are either actions or subgoals. A plan can be executed if its context condition holds, the precondition of each of its action steps holds when the step is reached, and each of its subgoal steps has an executable plan when the subgoal is reached. Note that, for many agent plans, it is only necessary that $B \models \chi$ for the plan to be executable, as the postcondition of an action step or achievement of a subgoal $s_i$ may establish the precondition of an action step $s_j$ or the context condition of plans for a subgoal step $s_j, i < j$. This is termed a $p$-effect (preparatory effect) in [Thangarajah et al., 2003].

A goal $g$ is considered achieved (and any intention with $g$ as top-level goal is dropped) if (and only if) all the steps in a plan $\pi_i$ for $g$ are successfully executed. We abuse notation slightly, and define the preconditions of a goal, $g$, as the union of the context conditions of the plans to achieve $g$, i.e., $\text{pre}(g) = \bigcup_{\pi_i} \text{pre}(\pi_i)$ (this is a technical device and only one of the plans for $g$ must be executable for $g$ to be “executable”). The postcondition of a goal, $g$, is $g$ itself, i.e., $g = \text{post}(g)$. We denote by $P$ the set of plans comprising the agent’s program.

3 Partially-Ordered Goal-Plan Trees

In much of the work on intention progression in BDI agents, the relationships between the plans, actions and subgoals that can be used to achieve a goal are represented by a hierarchical structure termed a goal-plan tree (GPT) [Thangarajah et al., 2003; Thangarajah and Padgham, 2011; Yao et al., 2016]. The root of a goal-plan tree is a goal-node representing a top-level goal, and its children are plan nodes representing the potential plans to achieve the top-level goal. The agent only needs to execute one of these plans to achieve the goal, hence, the goal nodes are viewed as or-nodes. The children of a plan node are the action and subgoal nodes corresponding to the steps in the plan body. The agent needs to execute all of the child nodes to achieve the goal. Thus, plan nodes are viewed as ordered and-nodes. Each subgoal node has its associated plans as children, giving rise to a tree structure representing all possible ways an agent can achieve the top-level goal.

While goal-plan trees have been shown to be effective for single-agent scheduling [Yao et al., 2016; Yao and Logan, 2016] and multi-agent scheduling where the program(s) of the other agents are known [Dann et al., 2020], they are less appropriate in multi-agent settings where the agents are not co-designed. We therefore propose a new approach to multi-agent intention progression in which agents schedule their intentions based on abstractions of their own programs which we call partially-ordered goal-plan trees (pGPT). A partially-ordered goal-plan tree is an alternating and-or tree that captures only the execution order implied by the $p$-effects in the agent’s program, rather than strictly adhering to the textual order of steps specified by the developer. More precisely:

**Definition 1 (Partially-Ordered Goal-Plan Tree)** A pGPT $T = (G, P, A, g_0, \text{children}, \{\prec_{\pi_i} : \pi_i \in P\})$ is an alternating and-or tree where:

- elements of $G \cup A$ are or-nodes (goals and actions\(^1\)) and elements of $P$ are and-nodes (plans);
- $g_0 \in G$ is the root (top-level goal);

\(^1\)Viewing action nodes as or-nodes without children is a technical device to simplify the definition.
• children : G ∪ P → 2^{G∪P∪A} is a function assigning a (non-empty) set of children to (non-leaf) nodes in the tree;
• or-nodes only have children in P: for g ∈ G, children(g) ⊆ P;
• and-nodes only have children in G ∪ A: for π ∈ P, children(π) ⊆ G ∪ A; and
• the set children(π) for each π ∈ P is partially ordered by ≺π.

An execution of a pGPT is a traversal of the tree starting at the root, g0. If visiting an or-node, the next step in the traversal is to pick any child and visit it; if in an and-node, visit each child in an order consistent with ≺π of the children. A pGPT T is executable if there is a traversal of T that is executable (given pre- and post conditions of plans and actions) from some initial state of the environment, i.e., if we start in this initial state and update it with the postconditions of each visited action, then the pre-conditions of the next action or context condition of the next plan in the traversal hold in the updated state.

A pGPT T for a top-level goal g0 and agent program Π = {π1, . . . , πn} can be built recursively as follows. Set g0 to be the root of the tree (or-node) and add as children(g0) and-nodes pi for each plan πi = g1 : c1 ← s1; . . . ; sm where gi = g0. For each plan and-node, pi, add as children(pi) or-nodes for the each of the steps of πi, s1; . . . ; sm. To establish the ordering ≺π, of the or-nodes representing the steps in πi, we proceed as follows:

• for each pair of steps s′i,j, 1 ≤ j < k ≤ m, if a literal l ∈ post(s′i,j) ∩ pre(s′k) then s′i,j ≺π, s′k, i.e., if s′i,j establishes a precondition for s′k, we add an ordering constraint that s′i,j must be executed before s′k (if s′k is a subgoal, before any step in whichever plan is selected to achieve s′k);
• if there is a step s′i, 1 ≤ c < j (or k < c ≤ m) where the complementary literal ∼ l ∈ post(s′i), we add s′c, ≺π, s′i to ≺π to ensure that the execution of s′i does not clutter l.

Finally, for each or-node where the corresponding step sj is a (sub)goal, gj, we recurse, i.e., we add as children(gj) and-nodes for each plan to achieve gj, and so on.

For example, given the agent program Π = {π0, π1, π2}:

π0 : g0 : χ0 ← g1; a0; a1
π1 : g1 : χ1 ← a2; a3; a4
π2 : g1 : χ2 ← a2

where χ0 = c0 ∧ c2, χ1 = c1, χ2 = c2, pre(a0) = c0, pre(a1) = c3 ∧ c4, pre(a2) = c2, pre(a3) = c1, pre(a4) = c1, pre(a5) = c6, post(a0) = c3, post(a1) = c5, post(a2) = c4, post(a3) = c6, post(a4) = c7, post(a5) = c8, we can generate the pGPT shown in Figure 1. In Figure 1, solid arrows connect goal nodes to the plan nodes that achieve the goal, e.g., p0 and p1 are two plan nodes to achieve the goal node g0; dashed arrows connect plan nodes to their execution steps, e.g., g1, a0, a1 are the execution steps of plan node p0; and double arrows indicate the partial ordering relationship between steps in a plan, e.g., action a0 must be executed before action a1, as one of the preconditions of a1 (c3) is established by the postcondition of a0.

4 Scheduling Approach

We now present a new approach to multi-agent intention scheduling based on pGPTs, which we call I_B. We adapt and extend Dann et al.’s [2020] Monte-Carlo Tree Search-based approach, I_A, which was shown to outperform several competing methods in a multi-agent setting. We first briefly recall the approach of Dann et al. before explaining how we adapt it to handle partial ordering.

4.1 Intention Scheduling with MCTS

The Monte Carlo Tree Search (MCTS) algorithm is a well-known heuristic search algorithm which has been shown to be successful in many multiplayer games [Brown et al., 2012]. The algorithm works by building a search tree incrementally through simulation. Starting at the root node, a tree policy is used to navigate to a leaf node. The tree policy strives to achieve a balance between exploration (traversing nodes that have rarely been visited) and exploitation (favouring steps that previously led to strong returns). The leaf node is then expanded, and a rollout policy is used to simulate steps until a terminal state is reached. Each node visited during the tree policy phase then has its average return updated based on the outcome of the simulation.

Early work on applying MCTS to intention scheduling [Yao et al., 2014; Yao et al., 2016b; Yao and Logan, 2016] focussed on the single-agent setting. More recently, Dann et al. [2020] extended the approach to multi-agent scheduling by introducing a payoff matrix, Pij. For each (i, j) pair, the payoff matrix captures the assumed payoffs that agent i receives upon the completion of agent j’s goals. For example, allied agents are modelled by defining positive payoffs for both the agent’s own goals and those of its allies, while adversarial agents are modelled by defining positive payoffs for the agent’s own goals but negative payoffs for the goals of other agents. In the scheduler’s underlying MCTS algorithm, the payoff matrix is used to calculate the return for each agent at the end of a rollout.
4.2 \( I_B \): Multi-Agent Scheduling with pGPTs

The scheduling approach taken in this work builds directly upon the \( I_A \) scheduler of Dann et al. [2020]. However, in contrast to \( I_A \), we do not assume that the order in which other agents will execute their plans is known. Rather, we assume that other agents may act in any way that is consistent with the ordering \( \prec \) implied by the p-effects in the agent’s own program. Consequently, we model agents’ intentions using pGPTs, rather than GPTs.

In the MCTS algorithm, this change impacts both node expansion and the rollout policy. Node expansion now adds branches for all steps that are executable according to the pGPT’s ordering constraints. Similarly, the rollout policy chooses uniformly from amongst all such steps. This growth in the branching factor is illustrated in Figure 2, which shows a pGPT (left) and one possible linearisation of it into a GPT (right). Suppose that nodes \( g_0 \) and \( p_0 \) have so far been traversed. In the (totally ordered) GPT, the only step now executable is \( a_0 \) (providing its pre-conditions hold). However, in the partially ordered pGPT, both \( a_0 \) and \( g_1 \) are executable.

An important consideration here is the amount of time taken to compute the set of executable steps, since this calculation is performed repeatedly during simulation. In Dann et al.’s [2020] GPT-based approach this is straightforward: since GPTs are totally ordered, one can just store a pointer to the next node in each GPT. However, in our pGPT-based approach, it is necessary to calculate the set of all steps that are executable according to the ordering constraints. This can be achieved by recursing through the tree structure, though such recursion is relatively expensive. Instead, we propose a more efficient method: First, for each node in each pGPT, we store a list of its unmet temporal dependencies, i.e. the set of steps that, according to \( \prec \), must be executed before the node in question. Then, each time a node is executed, we loop through its temporal dependencies (i.e., nodes that must come afterwards according to \( \prec \)) and update their lists of unmet dependencies. If all dependencies of a node are now met, it is added to a list of candidate steps. Finally, the candidate steps are filtered to ensure that all other logical requirements are met. In particular, to be executable, a candidate step must have its pre-conditions met. In addition, if the agent previously executed a plan or goal node, we force it to continue executing children of that node until it executes an action node. The rationale here is that it makes little sense for an agent to commit to a plan or goal until it has executed an underlying action. Pseudocode for the computation of candidate steps can be found in the extended version of the paper.

5 Evaluation

In this section, we evaluate our approach to multi-agent scheduling. We compare the performance of the \( I_B \) pGPT-based scheduler with \( I_A \), the GPT-based approach presented in [Dann et al., 2020], and random pGPT and GPT-based schedulers. As in [Dann et al., 2020], we consider three settings: fully aware, where agents know all the top-level goals of other agents (termed ‘full vision’ in [Dann et al., 2020]); partially aware, where agents know half of the other agents’ top-level goals (‘partial vision’); and unaware, where agents know none of the other agents’ top-level goals (‘naïve’). In each setting, the agents are assumed to have plans for the top-level goals they are aware of. However the \( I_A \) agents schedule on the assumption that the other agents have the same plans (and hence the same GPTs) for these goals as themselves. That is, unlike [Dann et al., 2020], the plans/GPTs of the other agents are not available to the agents; the other agents are essentially ‘black boxes’.

As in [Dann et al., 2020], we assume that the agents share the same model of actions (i.e., an action has the same pre- and postconditions for each agent). We also assume that each agent has the same set of actions available. While this is not true in all cases, it holds for a large class of applications, e.g., where the agents interact with their environment via an API.

While pGPTs can be derived from agent programs as explained in Section 3, in the interests of generality and simplicity, we implemented a pGPT generator that is capable of generating ‘synthetic’ pGPTs corresponding to agent programs of varying complexity (and hence multi-agent scheduling problems of varying difficulty). The pGPT generator is based on the GPT generator developed for the Intention Progression Competition.² (A detailed description of the pGPT generator can be found in the extended version of the paper.)

To generate the GPTs used by the \( I_A \) agents, for each \( I_A \) agent in each trial we generated a random linearisation of the pGPT used by the pGPT-based agent, similarly to Figure 2. Different linearisations will typically achieve subgoals and execute steps in a different order, but each ordering is consistent with \( \prec \). Each linearisation thus corresponds to a GPT derived from a valid program for the task, e.g., written by a different agent developer. All agents are therefore effectively using different programs for a given top-level goal, but while the pGPT agents abstract their programs into pGPTs, the \( I_A \) agents schedule and predict the next steps of other agents using the GPTs corresponding to the their own programs.

The random pGPT and GPT-based schedulers behave identically to the MCTS rollout policies used by \( I_B \) and \( I_A \), respectively. That is, they select uniformly from the set of all executable steps, \( \mathcal{E} \). The only difference between the pGPT and GPT-based schedulers is that, for the GPT-based scheduler, \( \mathcal{E} \) is calculated via a linearisation of the agent’s pGPTs, similarly to \( I_A \), and thus its behaviour is more restricted.

The generated pGPTs had a depth of 5. Each subgoal had two corresponding plans, with each plan containing three actions and one subgoal (except the lowest-level plans, which contained three actions only). For each trial, 12 GPTs were generated, with 6 assigned to each agent. Thus, for example,

```
Figure 2: A pGPT (left) and one possible linearisation (right).
```

²Available from intentionprogression.org.
the partially-aware $I_B$ schedulers had access to 9 pGPTs (6 for their own goals and 3 for the goals of the other agent). The partially-aware $I_A$ schedulers likewise had access to 9 GPTs, but the 3 corresponding to the goals of the other agent were derived from their own programs for those goals. The full set of pGPTs contained 80 environment variables, where this figure was chosen to yield enough scheduling clashes that the tasks were non-trivial, yet not excessively difficult.

We evaluated our approach under the three multi-agent settings considered by Dann et al. [2020]: allied, neutral and adversarial. In each of these settings there are two agents. The aim in the allied setting is to maximise the total number of team goals achieved. In the neutral setting, the aim is to maximise the agent’s own goal attainment while minimising the number of goals achieved by the external agent, i.e. to act selfishly. In the adversarial setting, the aim is to maximise the agent’s own goal attainment while minimising the number of goals achieved by the external agent. Accordingly, we use the following scoring methodology:

- **Allied**: we report $own\_goals + ally\_goals$. Each agent is assigned 6 pGPTs, so the possible score range is $[0, 12]$.
- **Neutral**: we report $own\_goals$ only, so the possible score range is $[0, 6]$.
- **Adversarial**: we report $own\_goals – opponent\_goals$, so the possible score range is $[-6, 6]$.

The results of these experiments are provided in Tables 1, 2 and 3, with scores averaged over 500 randomly generated pGPT forests. Cell values indicate the score achieved by the agent from that row when partnered with the agent above. The numeric suffixes to the MCTS-based agents’ names indicate their level of awareness of the other agent’s goals ($100 = $fully aware, $50 = $partially aware, $0 = $unaware).

A striking feature of the results is their consistency: In each setting and for each partner agent type, the fully aware $I_B$ scheduler performed best. Given that $I_B$ is more flexible than $I_A$, both in terms of its ability to act and its ability to model the behaviour of other agents, this may seem unsurprising. However, recall from Section 4.2 that the MCTS search tree has a greater branching factor under $I_B$ than under $I_A$. Since $I_A$ and $I_B$ were afforded the same number of simulated rollouts this means that the rollouts were spread more thinly under $I_B$, with nodes’ average returns calculated from fewer samples. $I_B$’s consistent edge over $I_A$ shows that this trade-off was worthwhile, i.e., $I_B$’s greater uncertainty about the return was more than compensated for by its extra flexibility.

Note that the significance of certain results is best appreciated by considering the number of goals unachieved. For example, in the allied setting, the ($I_{A,100}, I_{A,100}$) partnership averaged 10.157 goals, while ($I_{B,100}, I_{B,100}$) averaged 11.475; an increase of only $\approx 1.3$ goals. However, since the best possible score in this setting is 12, this represents a $72\%$ reduction in unachieved goals.

The unaware agents $I_{A,0}$ and $I_{B,0}$ have no knowledge of other agents’ goals or the plans used to achieve those goals, and so are unable to make any predictions about the next action of the other agent. As such, they are essentially doing single-agent scheduling with no consideration of the other agents’ intentions, which is the baseline case in [Dann et al., 2020]. The small improvement in performance of $I_{B,0}$ over $I_{A,0}$ in the allied and neutral settings ($4\%$ increase in goals achieved by two $I_{B,0}$ agents over two $I_{A,0}$ agents in the allied setting, and $3\%$ in neutral) is attributable to the $I_B$ agents having more flexibility in the order in which actions are exe-

### Table 1: Allied setting. The best total team score with each ally type is bolded.

<table>
<thead>
<tr>
<th>Total Score</th>
<th>rand_pGPT</th>
<th>rand_GPT</th>
<th>$I_{B,100}$</th>
<th>$I_{B,50}$</th>
<th>$I_{B,0}$</th>
<th>$I_{A,100}$</th>
<th>$I_{A,50}$</th>
<th>$I_{A,0}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{B,100}$</td>
<td>9.103</td>
<td>8.327</td>
<td>11.475</td>
<td>11.158</td>
<td>10.779</td>
<td>10.904</td>
<td>10.661</td>
<td>10.247</td>
</tr>
<tr>
<td>$I_{B,0}$</td>
<td>5.676</td>
<td>5.850</td>
<td>10.779</td>
<td>9.342</td>
<td>7.180</td>
<td>9.522</td>
<td>8.673</td>
<td>7.097</td>
</tr>
<tr>
<td>$I_{A,0}$</td>
<td>5.631</td>
<td>5.416</td>
<td>10.247</td>
<td>9.204</td>
<td>7.097</td>
<td>9.357</td>
<td>8.466</td>
<td>8.977</td>
</tr>
</tbody>
</table>

### Table 2: Neutral setting. The best results achieved with respect to the other scheduler type are bolded.

<table>
<thead>
<tr>
<th>Score</th>
<th>rand_pGPT</th>
<th>rand_GPT</th>
<th>$I_{B,100}$</th>
<th>$I_{B,50}$</th>
<th>$I_{B,0}$</th>
<th>$I_{A,100}$</th>
<th>$I_{A,50}$</th>
<th>$I_{A,0}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{B,100}$</td>
<td>5.537</td>
<td>5.211</td>
<td>5.339</td>
<td>5.299</td>
<td>5.548</td>
<td>5.098</td>
<td>5.098</td>
<td>5.282</td>
</tr>
<tr>
<td>$I_{B,0}$</td>
<td>3.738</td>
<td>3.907</td>
<td>3.510</td>
<td>3.475</td>
<td>3.585</td>
<td>3.621</td>
<td>3.592</td>
<td>3.787</td>
</tr>
</tbody>
</table>
The experiments also show that a key result from [Dann et al., 2016c] holds in the black-box setting; namely, both $I_A$ and $I_B$ improve as they are afforded more knowledge of the other agent’s goals, demonstrating that they truly are intention-aware. In the extended version of the paper, we break the allied setting scores down into own_goals and ally_goals. From this, it is evident that the fully aware $I_B$ scheduler was the most effective at helping its partner achieve goals, similar to findings in [Dann et al., 2020].

## 6 Related Work

The first MCTS-based approach to intention scheduling with BDI agents was that of Yao et al. [2014]. They used a variant of MCTS called Single-Player MCTS [Schadd et al., 2012] to schedule the intentions of a single agent at the level of plans rather than actions. The work was later extended to scheduling intentions at the action level [Yao and Logan, 2016], with deadlines [Yao et al., 2016b] and for exploiting synergies [Yao et al., 2016c]. This work on single agent scheduling formed the basis for the work by Dann et al. [2020] on using MCTS-based scheduling for multi-agent settings.

Another line of work considers incorporating an HTN planner into a BDI agent language, thus providing the ability to look-ahead and find solutions to ensure the successful execution of an intention, when necessary [Sardina et al., 2006; de Silva, 2017]. Whilst these methods focus on finding solutions to achieve a single intention and do not take into account interactions with other concurrent intentions, it could be possible to incorporate the work of Clement et al., mentioned above, to centrally schedule intentions of BDI agents.

Conversely, de Silva [2018] considers incorporating ideas from the BDI paradigm into HTN planning, so as to support interleaved deliberation, acting, and failure recovery. Again, however, this work focuses only on the pursuit of single goals.

Finally, we note the coverage-based approach as used in [Thangarajah et al., 2012; Waters et al., 2014; Waters et al., 2015], where the probability of executing plans to achieve a goal in the different possible states of the environment is used to schedule intentions. The intuition is to progress the intention that has the highest probability of becoming non-executable in future states. In contrast to our work, this approach has so far been applied only to single-agent settings.

Beyond the differences already mentioned, unlike our approach, all of the above methods require full knowledge of the agents’ intention structures.

## 7 Conclusion

In this paper we proposed an approach to multi-agent intention progression in environments where the other agents are not co-designed, and thus their precise program(s) are unknown. We evaluated our approach in allied, neutral and adversarial settings, and showed that it outperformed MCTS-based scheduling where agents assume that other agents will execute plans in the same order as themselves. A possible direction for future work is to further relax the assumptions regarding other agents, e.g. by assuming that other agents will pursue some subset of all known goals in the environment, but where this subset must be inferred at runtime.

## Acknowledgments

Yuan Yao was supported by National Natural Science Foundation of China (61906169) and Zhejiang Provincial Natural Science Foundation of China (LQ19F030009).
References


