Multi-Agent Intention Progression with Black-Box Agents

Michael Dann[1*], **Yuan Yao**[2], **Brian Logan**[3] and **John Thangarajah**[1]
[1]RMIT University
[2]Zhejiang University of Technology
[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.*, 2016c] 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 "egocentric" 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 differently. In this paper, we propose a new approach to in-

*Contact Author. Source code for our experiments is available at: https://github.com/mchldann/pGPT_IJCAI.

tention progression in a multi-agent setting, in which agents schedule their intentions and predict the actions of other agents based on an abstraction of their own program called a partially-ordered goal-plan tree (pGPT). A *partially-ordered goal-plan tree* is an alternating and-or tree that captures only the execution order implied by the dependency relationships inherent in the agent's program. We show how a pGPT can be derived from the program of a BDI agent, and how the MCTS-based approach in [Dann *et al.*, 2020] can be extended and adapted to schedule an agent's intentions using pGPTs in multi-agent settings. As in [Dann *et al.*, 2020], we evaluate our approach in cooperative, selfish and adversarial multi-agent settings. Our results indicate that pGPT-based scheduling out-performs MCTS-based scheduling where agents assume that other agents behave in the same way as themselves.

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 such that $p, \neg p \in B$. 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 $\phi = pre(a_i)$ that must be true before the execution of the action, and the *postconditions* of the action are a set of literals $\psi = post(a_i)$ 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 π_i is of the form $g : \chi \leftarrow s_1; \ldots; s_m$, where $\chi = pre(\pi_i)$ is a set of literals specifying the *context condition* which must be true for π_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 π_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., $pre(g) = \bigcup_{\pi_i} pre(\pi_i)$ (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., $g = post(g)$. 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.*, 2016a]. 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.*, 2016c; 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 $T = (G, P, A, g_0, children, \{\prec_\pi : \pi \in P\})$ 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);*
- *$children : G \cup P \longrightarrow 2^{G \cup P \cup A}$ is a function assigning a (non-empty) set of children to (non-leaf) nodes in the tree;*

[1] Viewing action nodes as or-nodes without children is a technical device to simplify the definition.

- *or-nodes only have children in P: for $g \in G$, $children(g) \subseteq P$;*
- *and-nodes only have children in $G \cup A$: for $\pi \in P, children(\pi) \subseteq G \cup A$; and*
- *the set $children(\pi)$ for each $\pi \in P$ is partially ordered by $\prec_\pi$.*

An execution of a pGPT is a traversal of the tree starting at the root, g_0. 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 $\prec_\pi$ 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 g_0 and agent program $\Pi = \{\pi_1, \ldots, \pi_n\}$ can be built recursively as follows. Set g_0 to be the root of the tree (or-node) and add as $children(g_0)$ and-nodes p_i for each plan $\pi_i = g_i : \chi_i \leftarrow s^i_1; \ldots; s^i_m$ where $g_i = g_0$. For each plan and-node, p_i, add as $children(p_i)$ or-nodes for the each of the steps of π_i, $s^i_1; \ldots; s^i_m$. To establish the ordering $\prec_{\pi_i}$ of the or-nodes representing the steps in π_i, we proceed as follows:

- for each pair of steps $s^i_j, s^i_k, 1 \leq j < k \leq m$, if a literal $l \in post(s^i_j) \cap pre(s^i_k)$ then $s^i_j \prec_{\pi_i} s^i_k$, i.e., if s^i_j establishes a precondition for s^i_k, we add an ordering constraint that s^i_j must be executed before s^i_k (if s^i_k is a subgoal, before any step in whichever plan is selected to achieve s^i_k);
- if there is a step $s^i_c, 1 \leq c < j$ (or $k < c \leq m$) where the complementary literal $\sim l \in post(s^i_c)$, we add $s^i_c \prec_{\pi_i} s^i_j$ (respectively $s^i_k \prec_{\pi_i} s^i_c$) to $\prec_{\pi_i}$ to ensure that the execution of s^i_c does not clobber l.

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

For example, given the agent program $\Pi = \{\pi_0, \pi_1, \pi_2\}$:

$$\pi_0 : g_0 : \chi_0 \leftarrow g_1; a_0; a_1$$
$$\pi_1 : g_0 : \chi_1 \leftarrow a_3; a_4; a_5$$
$$\pi_2 : g_1 : \chi_2 \leftarrow a_2$$

where $\chi_0 = c_0 \wedge c_2$, $\chi_1 = c_1$, $\chi_2 = c_2$, $pre(a_0) = c_0$, $pre(a_1) = c_3 \wedge c_4$, $pre(a_2) = c_2$, $pre(a_3) = c_1$, $pre(a_4) = c_1$, $pre(a_5) = c_6$, $post(a_0) = c_3$, $post(a_1) = c_5$, $post(a_2) = c_4$, $post(a_3) = c_6$, $post(a_4) = c_7$, $post(a_5) = c_8$, 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., p_0 and p_1 are two plan nodes to achieve the goal node g_0; dashed arrows connect plan nodes to their execution steps, e.g., g_1, a_0, a_1 are the execution steps of plan node p_0; and double arrows indicate the partial ordering relationship between steps in a plan, e.g., action a_0 must be executed before action a_1, as one of the preconditions of a_1 (c_3) is established by the postcondition of a_0.

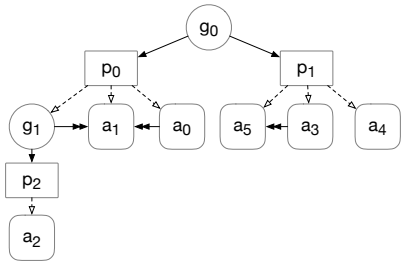

Figure 1: An example of a pGPT.

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 [Browne *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.*, 2016c; 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, $\mathcal{P}_{ij}$. 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 dependents* (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 Appendix.

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'.

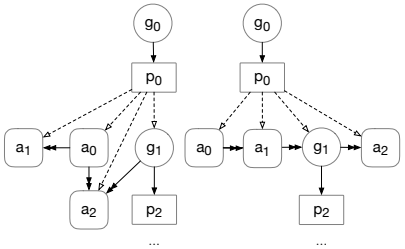

Figure 2: A pGPT (left) and one possible linearisation (right).

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 Appendix.)

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, 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

[2] Available from intentionprogression.org.

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 and disregard 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 *un*achieved. 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 ≈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 executed. This allows each I_B agent to avoid more conflicts between its own intentions. However, neither the I_B_0 nor the I_A_0 agents can effectively avoid conflicts with the intentions of the other agent (since these are unknown). As the results show, as the agents' knowledge of the other agent's goals increases, the performance of both I_B and I_A increases, but I_B is better able to exploit the additional information. For exam-

		Ally							
		rand_pGPT	rand_GPT	I_B_100	I_B_50	I_B_0	I_A_100	I_A_50	I_A_0
Total Score	rand_pGPT	4.023	3.771	9.103	7.743	5.766	7.744	6.864	5.631
	rand_GPT	3.771	3.634	8.327	7.400	5.850	7.292	6.649	5.416
	I_B_100	**9.103**	**8.327**	**11.475**	**11.158**	**10.779**	**10.904**	**10.661**	**10.247**
	I_B_50	7.743	7.400	11.158	10.446	9.342	10.342	9.895	9.204
	I_B_0	5.766	5.850	10.779	9.342	7.180	9.522	8.673	7.097
	I_A_100	7.744	7.292	10.904	10.342	9.522	10.157	9.928	9.357
	I_A_50	6.864	6.649	10.661	9.895	8.673	9.928	9.343	8.466
	I_A_0	5.631	5.416	10.247	9.204	7.097	9.357	8.466	6.897

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

		Other Scheduler							
		rand_pGPT	rand_GPT	I_B_100	I_B_50	I_B_0	I_A_100	I_A_50	I_A_0
Score	rand_pGPT	2.001	2.070	2.034	1.989	2.017	2.029	2.000	2.102
	rand_GPT	1.723	1.866	1.793	1.781	1.830	1.840	1.883	1.863
	I_B_100	**5.537**	**5.211**	**5.339**	**5.299**	**5.548**	**5.098**	**5.098**	**5.282**
	I_B_50	4.677	4.631	4.510	4.419	4.621	4.452	4.468	4.568
	I_B_0	3.738	3.907	3.510	3.475	3.585	3.621	3.592	3.787
	I_A_100	4.577	4.538	4.418	4.355	4.643	4.455	4.418	4.562
	I_A_50	4.007	4.171	3.898	3.795	4.025	3.945	3.852	4.175
	I_A_0	3.457	3.669	3.319	3.115	3.235	3.282	3.296	3.479

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

Net Score	Opponent							
	rand_pGPT	rand_GPT	I_B_100	I_B_50	I_B_0	I_A_100	I_A_50	I_A_0
rand_pGPT	0.000	0.224	-4.839	-3.001	-1.767	-3.429	-2.274	-1.270
rand_GPT	-0.224	0.000	-4.205	-2.951	-2.094	-3.395	-2.605	-1.690
I_B_100	**4.839**	**4.205**	**0.000**	**2.280**	**4.317**	**1.666**	**2.542**	**3.735**
I_B_50	3.001	2.951	-2.280	0.000	2.050	-0.471	0.746	2.058
I_B_0	1.767	2.094	-4.317	-2.050	0.000	-2.604	-0.977	0.652
I_A_100	3.429	3.395	-1.666	0.471	2.604	0.000	1.280	2.672
I_A_50	2.274	2.605	-2.542	-0.746	0.977	-1.280	0.000	1.489
I_A_0	1.270	1.690	-3.735	-2.058	-0.652	-2.672	-1.489	0.000

Table 3: Adversarial setting. The best net score (own goals minus opponent goals) achieved against each opponent type is bolded.

ple, in the neutral setting, two I_B_100 agents can achieve 20% more goals than two I_A_100 agents because they are better at avoiding conflicts with the other agent's intentions. Indeed the total number of goals achieved by two I_B_100 agents in the neutral setting is greater than two I_A_100 agents in the allied setting, i.e., two 'selfish' I_B_100 agents are more effective than two cooperating I_A_100 agents. A similar pattern is seen when I_B is paired with other schedulers.

The experiments also show that a key result from [Dann *et al.*, 2020] 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 Appendix, 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.

There have been other approaches to scheduling intentions. For example, the early work by Thangarajah et al. [2002; 2003; 2011] and Clement et al. [2007; 1999; 2000] used the notion of summary information to look-ahead at the intention structures and schedule accordingly. Whilst Thangarajah et al. focussed on scheduling the intentions of a single agent in BDI setting using GPTs, Clement et al. focussed on centrally scheduling the intentions of multiple agents in planning agents using hierarchical task networks (HTNs).

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 [Sardiña *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

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on*

Computational Intelligence and AI in games, 4(1):1–43, 2012.

[Clement and Durfee, 1999] Bradley J. Clement and Edmund H. Durfee. Theory for coordinating concurrent hierarchical planning agents using summary information. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI 1999)*, pages 495–502, 1999.

[Clement and Durfee, 2000] Bradley J. Clement and Edmund H. Durfee. Performance of coordinating concurrent hierarchical planning agents using summary information. In *Proceedings of the 4th International Conference on Multi-Agent Systems*, pages 373–374, 2000.

[Clement *et al.*, 2007] Bradley J. Clement, Edmund H. Durfee, and Anthony C. Barrett. Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research*, 28:453–515, 2007.

[Dann *et al.*, 2020] Michael Dann, John Thangarajah, Yuan Yao, and Brian Logan. Intention-aware multiagent scheduling. In B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, and G. Sukthankar, editors, *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020)*, pages 285–293, 2020.

[de Silva, 2017] Lavindra de Silva. BDI agent reasoning with guidance from HTN recipes. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2017)*, pages 759–767, 2017.

[de Silva, 2018] Lavindra de Silva. HTN acting: A formalism and an algorithm. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018)*, pages 363–371, 2018.

[Logan *et al.*, 2017] Brian Logan, John Thangarajah, and Neil Yorke-Smith. Progressing intention progresson: A call for a goal-plan tree contest. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)*, pages 768–772, 2017.

[Rao and Georgeff, 1992] Anand S. Rao and Michael P. Georgeff. An abstract architecture for rational agents. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR 1992)*, pages 439–449, 1992.

[Sardiña *et al.*, 2006] Sebastian Sardiña, Lavindra de Silva, and Lin Padgham. Hierarchical planning in BDI agent programming languages: a formal approach. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 1001–1008, 2006.

[Schadd *et al.*, 2012] Maarten P. D. Schadd, Mark H. M. Winands, Mandy J. W. Tak, and Jos W. H. M. Uiterwijk. Single-player Monte-Carlo tree search for SameGame. *Knowledge-Based Systems*, 34:3–11, 2012.

[Thangarajah and Padgham, 2011] John Thangarajah and Lin Padgham. Computationally effective reasoning about goal interactions. *Journal of Automated Reasoning*, 47(1):17–56, 2011.

[Thangarajah *et al.*, 2002] John Thangarajah, Michael Winikoff, Lin Padgham, and Klaus Fischer. Avoiding resource conflicts in intelligent agents. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, pages 18–22, 2002.

[Thangarajah *et al.*, 2003] John Thangarajah, Lin Padgham, and Michael Winikoff. Detecting & avoiding interference between goals in intelligent agents. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 721–726, 2003.

[Thangarajah *et al.*, 2012] John Thangarajah, Sebastian Sardina, and Lin Padgham. Measuring plan coverage and overlap for agent reasoning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, pages 1049–1056, 2012.

[Waters *et al.*, 2014] Max Waters, Lin Padgham, and Sebastian Sardina. Evaluating coverage based intention selection. In *Proceedings of the 13th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2014)*, pages 957–964, 2014.

[Waters *et al.*, 2015] Max Waters, Lin Padgham, and Sebastian Sardiña. Improving domain-independent intention selection in BDI systems. *Autonomous Agents and Multi-Agent Systems*, 29(4):683–717, 2015.

[Yao and Logan, 2016] Yuan Yao and Brian Logan. Action-level intention selection for BDI agents. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems*, pages 1227–1236, 2016.

[Yao *et al.*, 2014] Yuan Yao, Brian Logan, and John Thangarajah. SP-MCTS-based intention scheduling for BDI agents. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 2014.

[Yao *et al.*, 2016a] Yuan Yao, Lavindra de Silva, and Brian Logan. Reasoning about the executability of goal-plan trees. In *Proceedings of the 4th International Workshop on Engineering Multi-Agent Systems (EMAS 2016)*, pages 181–196, Singapore, May 2016.

[Yao *et al.*, 2016b] Yuan Yao, Brian Logan, and John Thangarajah. Intention selection with deadlines. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI-2016)*, pages 1700–1701, 2016.

[Yao *et al.*, 2016c] Yuan Yao, Brian Logan, and John Thangarajah. Robust execution of BDI agent programs by exploiting synergies between intentions. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016)*, pages 2558–2565, 2016.

A Appendix

In this appendix we provide some further experimental details that did not fit within the space constraints of the main text.

A.1 Further Scheduler Details

As in [Dann *et al.*, 2020], we assume that the environment is turn-based, with each turn persisting until an agent selects an action node from one of its pGPTs. Likewise, we assume that agents can pass their turn, either by choice or necessity (if there are no executable steps left).

The MCTS-based schedulers are afforded a simulation budget per *turn*, meaning that they must select an action node within this budget. Accordingly, if the first greedy step in the search tree does not lead to an action node, we keep following the chain of greedy steps until an action node is encountered. For example, the scheduler might select goal g_0, then the underlying plan p_0, then finally the action a_0 from within p_0.

The computational budget for the MCTS-based schedulers is specified via two parameters: α, the number of node expansions to be performed; and β, the number of simulations performed per node expansion. Following previous work [Yao and Logan, 2016; Dann *et al.*, 2020], we set $\alpha = 100$ and $\beta = 10$.

In tree policy phase of MCTS, step selections are determined by maximising the *UCB1* value [Browne *et al.*, 2012], which sums the average return with an exploration bonus:

$$\text{UCB1 value} = \frac{sum(\text{payoffs})}{n} + c\sqrt{\frac{2\ln N}{n}}$$

where c controls the scale of the exploration bonus, n is the number of times the step has been tried, and N is the total number of rollouts performed so far. Following [Dann *et al.*, 2020], we set $c = \sqrt{2}$.

Also like [Dann *et al.*, 2020], we ran "mirror matches" for each generated set of pGPTs. In the second leg of each mirror match, the 6 goals assigned to each agent were swapped, as well as the first agent to move. This was done to reduce noise due to unfair task assignments. All results were averaged over 500 mirror matches on different sets of generated pGPTs.

One subtlety regarding the payoff matrix used by both I_A and I_B is that it specifies not only the overall objective for the scheduler itself, but also captures assumptions about the objectives of the external agents. In the adversarial setting, it is reasonable to assume that external agents will also behave adversarially. However, in certain real-world tasks (e.g. driving a car in heavy traffic), it can be difficult to know whether external agents will behave in a friendly manner or selfishly. Dann et al. [2020] found in their allied and neutral experiments that I_A was impacted little by switching the friendly/selfish assumption about other agents. Therefore, to avoid having an excessive number of agent variants, we configured the payoff matrix to assume neutral behaviour from external agents in both settings.

A.2 Algorithm for Calculating Candidate Steps

In Section 4.2 of the main text we describe an algorithm for computing the set of executable steps in a pGPT efficiently. The pseudocode for this approach, including how it fits into the main environment loop, is provided in Algorithm 1.

A.3 GPG Generator

We have proposed a pGPT generator based on the existing GPT generator used in the *Intention Progression Competition*[3]. As with the original GPT generator, developers can specify the input parameters to control the shape and the properties of the pGPT, e.g., the depth of the tree, the number of plans to achieve a goal and so on. To generate a pGPT, we need first generate a GPT as before, and then replace the total ordering relationship of steps in each plan by partial ordering constraints implied by the p-effects of these steps as described in Section 3. Each action node and goal node maintains a set of action and goal nodes named prerequisite steps which must be executed or achieved before the action or subgoal itself. From the top-level goal, we systematically check execution steps in each plan to achieve the top-level goal and compute all p-effects in these plans. If there is a p-effect c established by a step s_i (i.e., c is one of s_i's postcondition) and it is one of the preconditions of a step s_j (s_i and s_j belong to the same plan), then we add s_i to the set of prerequisite steps of s_j. Moreover, any steps s_k that establish $\neg c$ must be executed before s_i or after s_j, i.e., s_k is either in the set of prerequisite steps of s_i or s_j is one of s_k's prerequisite steps. We then recurse the above procedure for subgoals in the plan to compute all the dependency relationships for the hierarchies below. Finally, an XML file representing the pGPT is generated.

A.4 Breakdown of *Allied* and *Adversarial* Results

In the main text, we report combined team scores for the allied setting and score differentials for the adversarial setting. In Tables 4 and 5, we break these figures down into individual agent scores so that the effect of each scheduling approach on the partnered agent can be seen more clearly.

[3] https://www.intentionprogression.org/

Algorithm 1 Environment Loop with Candidate Steps

```
1:  initialise environment state, env
2:  initialise a list of unmet temporal dependencies for each node.
3:  C ← G        ▷ initialise candidate steps to set of top-level goals
4:
5:  while there are executable steps remaining do
6:      E ← FILTERCANDIDATES(C)        ▷ get executable steps
7:      s ← agent.SCHEDULESTEP(E)
8:      update env according to s
9:      for each temporal dependent d of s do
10:         Remove s from d's list of unmet temporal dependencies.
11:         if all temporal dependencies of d are now met then
12:             C ← C ∪ d
13:
14: function FILTERCANDIDATES(C)
15:     E ← ∅
16:     for each candidate step c ∈ C do
17:         if all pre-conditions of c are met
                and c conforms to any previous goal/plan selection
                then E ← E ∪ c
18:     return E
```

		Ally							
		rand_pGPT	rand_GPT	I_B_100	I_B_50	I_B_0	I_A_100	I_A_50	I_A_0
Total Score	rand_pGPT	4.023	3.771	9.103	7.743	5.766	7.744	6.864	5.631
	rand_GPT	3.771	3.634	8.327	7.400	5.850	7.292	6.649	5.416
	I_B_100	**9.103**	**8.327**	**11.475**	**11.158**	**10.779**	**10.904**	**10.661**	**10.247**
	I_B_50	7.743	7.400	11.158	10.446	9.342	10.342	9.895	9.204
	I_B_0	5.766	5.850	10.779	9.342	7.180	9.522	8.673	7.097
	I_A_100	7.744	7.292	10.904	10.342	9.522	10.157	9.928	9.357
	I_A_50	6.864	6.649	10.661	9.895	8.673	9.928	9.343	8.466
	I_A_0	5.631	5.416	10.247	9.204	7.097	9.357	8.466	6.897
Own Goals	rand_pGPT	2.012	2.076	3.950	3.095	2.008	3.525	2.908	2.119
	rand_GPT	1.695	1.817	3.493	2.769	1.826	3.205	2.624	1.831
	I_B_100	5.154	4.835	5.738	5.340	5.038	5.532	5.183	4.833
	I_B_50	4.648	4.631	5.818	5.223	4.618	5.602	5.132	4.721
	I_B_0	3.758	4.023	5.741	4.723	3.590	5.486	4.728	3.829
	I_A_100	4.219	4.087	5.372	4.740	4.037	5.078	4.682	4.128
	I_A_50	3.955	4.025	5.478	4.762	3.944	5.246	4.672	4.014
	I_A_0	3.513	3.586	5.414	4.483	3.268	5.229	4.451	3.448
Ally Goals	rand_pGPT	2.012	1.695	5.154	4.648	3.758	4.219	3.955	3.513
	rand_GPT	2.076	1.817	4.835	4.631	4.023	4.087	4.025	3.586
	I_B_100	3.950	3.493	5.738	5.818	5.741	5.372	5.478	5.414
	I_B_50	3.095	2.769	5.340	5.223	4.723	4.740	4.762	4.483
	I_B_0	2.008	1.826	5.038	4.618	3.590	4.037	3.944	3.268
	I_A_100	3.525	3.205	5.532	5.602	5.486	5.078	5.246	5.229
	I_A_50	2.908	2.624	5.183	5.132	4.728	4.682	4.672	4.451
	I_A_0	2.119	1.831	4.833	4.721	3.829	4.128	4.014	3.448

Table 4: A breakdown of the results for the allied setting. The best total team score with each ally type is bolded. Since each scheduler was assigned 6 GPTs, the maximum possible team score was 12.

		Opponent							
		rand_pGPT	rand_GPT	I_B_100	I_B_50	I_B_0	I_A_100	I_A_50	I_A_0
Net Score	rand_pGPT	0.000	0.224	-4.839	-3.001	-1.767	-3.429	-2.274	-1.270
	rand_GPT	-0.224	0.000	-4.205	-2.951	-2.094	-3.395	-2.605	-1.690
	I_B_100	**4.839**	**4.205**	**0.000**	**2.280**	**4.317**	**1.666**	**2.542**	**3.735**
	I_B_50	3.001	2.951	-2.280	0.000	2.050	-0.471	0.746	2.058
	I_B_0	1.767	2.094	-4.317	-2.050	0.000	-2.604	-0.977	0.652
	I_A_100	3.429	3.395	-1.666	0.471	2.604	0.000	1.280	2.672
	I_A_50	2.274	2.605	-2.542	-0.746	0.977	-1.280	0.000	1.489
	I_A_0	1.270	1.690	-3.735	-2.058	-0.652	-2.672	-1.489	0.000
Own Goals	rand_pGPT	2.010	2.049	0.319	1.377	2.035	0.721	1.438	2.119
	rand_GPT	1.825	1.878	0.412	1.238	1.895	0.636	1.298	1.876
	I_B_100	5.158	4.617	2.215	3.627	5.039	3.160	3.716	4.576
	I_B_50	4.379	4.189	1.348	2.872	4.309	2.301	3.085	4.101
	I_B_0	3.801	3.989	0.723	2.259	3.600	1.479	2.669	3.856
	I_A_100	4.150	4.031	1.494	2.772	4.082	2.179	3.026	4.005
	I_A_50	3.712	3.902	1.174	2.339	3.647	1.746	2.709	3.785
	I_A_0	3.389	3.567	0.841	2.043	3.204	1.333	2.295	3.480
Opponent Goals	rand_pGPT	2.010	1.825	5.158	4.379	3.801	4.150	3.712	3.389
	rand_GPT	2.049	1.878	4.617	4.189	3.989	4.031	3.902	3.567
	I_B_100	0.319	0.412	2.215	1.348	0.723	1.494	1.174	0.841
	I_B_50	1.377	1.238	3.627	2.872	2.259	2.772	2.339	2.043
	I_B_0	2.035	1.895	5.039	4.309	3.600	4.082	3.647	3.204
	I_A_100	0.721	0.636	3.160	2.301	1.479	2.179	1.746	1.333
	I_A_50	1.438	1.298	3.716	3.085	2.669	3.026	2.709	2.295
	I_A_0	2.119	1.876	4.576	4.101	3.856	4.005	3.785	3.480

Table 5: A breakdown of the results for the adversarial setting. The best net score (own goals minus opponent goals) achieved against each opponent type is bolded.