Multi-Agent Intention Progression with Reward Machines

Michael Dann[1] , **Yuan Yao**[2] , **Natasha Alechina**[3] , **Brian Logan**[3,4] and **John Thangarajah**[1]
[1]RMIT University
[2]University of Nottingham, Ningbo China
[3]Utrecht University
[4] University of Aberdeen
{michael.dann, john.thangarajah}@rmit.edu.au, yuan.yao@nottingham.edu.cn,
{n.a.alechina, b.s.logan}@uu.nl

Abstract

Recent work in multi-agent intention scheduling has shown that enabling agents to predict the actions of other agents when choosing their own actions may be beneficial. However existing approaches to 'intention-aware' scheduling assume that the programs of other agents are known, or are "similar" to that of the agent making the prediction. While this assumption is reasonable in some circumstances, it is less plausible when the agents are not co-designed. In this paper, we present a new approach to multi-agent intention scheduling in which agents predict the actions of other agents based on a high-level specification of the *tasks* performed by an agent in the form of a *reward machine* (RM) rather than on its (assumed) program. We show how a reward machine can be used to generate tree and rollout policies for an MCTS-based scheduler. We evaluate our approach in a range of multi-agent environments, and show that RM-based scheduling out-performs previous intention-aware scheduling approaches in settings where agents are not co-designed.

1 Introduction

The Belief-Desire-Intention (BDI) model [Rao and Georgeff, 1992] forms the basis of much of the research on symbolic models of agency and agent-oriented software engineering [de Silva *et al.*, 2020]. In the BDI model, *beliefs* represent the agent's information about its environment, other agents and itself, *goals* (desires) are states of affairs to achieve, and *intentions* are commitments to achieving particular goals. A BDI agent program consists of a set of initial beliefs and a set of plans for achieving goals. Plans are composed of *primitive actions* that directly change the state of the environment, and *subgoals* which are achieved by their own plans. The BDI approach to agent development has a number of significant advantages compared to other approaches to developing rational agents. End users find programs couched in mentalistic terms such as 'beliefs' and 'goals' easier to understand [Norling and Ritter, 2004], and the use of predefined plans encoding standard responses to situations gives predictable and explainable behaviour that can be more easily validated by end-users and other stakeholders [Broekens *et al.*, 2010].

A key problem for a BDI agent with multiple intentions is to determine 'what to do next': which goal the agent should be trying to achieve, and which plan it should use to achieve it. This problem is termed the *intention progression problem* (IPP) [Logan *et al.*, 2017]. A number of approaches to various aspects of the IPP in the single agent setting 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.*, 2016b] approaches. There has been relatively little work on intention progression in 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., when the execution of a step in a plan of one agent makes the execution of a step in a plan of another agent impossible. Dann et al. [2020] extended the MCTS-based approach in [Yao and Logan, 2016] to a multi-agent setting, and showed that their 'intention-aware' scheduler I_A out-performed non-intention-aware scheduling such as [Yao and Logan, 2016]. However, their approach assumes that agents have access to the plans used by other agents to achieve their goals. More recently, Dann et al. [2021] proposed an approach based on *partially-ordered goal-plan trees*, in which agents schedule their intentions and predict the actions of other agents based on an abstraction of their own program. While their I_B scheduler does not require knowledge of the plans comprising the other agents' programs, there is no guarantee that the other agents in a multi-agent environment have "similar" programs, or are even BDI agents.

In this paper, we present I_{RM}, a new approach to multi-agent intention scheduling in which agents predict the actions of other agents based on a high-level specification of the *tasks* performed by an agent rather than its *program* (or assumed program). For many scenarios, it is reasonable to assume an agent knows the high-level specification of the tasks performed by other agents. For example, when agents cooperate to achieve a team goal, it is reasonable to assume that each agent knows the tasks assigned to the other agents

in the team, but may not know how they will achieve them. Tasks are assumed to be specified declaratively, for example, by Linear Time Temporal Logic (LTL) formulas. We focus on tasks represented by *reward machines* (RM) introduced in [Toro Icarte *et al.*, 2018]. Reward machines are Mealy machines (automata with outputs) which can be computed from task specifications expressed in a range of goal and property specification languages, including LTL and LTL$_f$ (LTL on finite traces), regular expressions and other goal specification languages [Camacho *et al.*, 2019]. Critically, RMs encode the logical structure of a task rather than a particular implementation of the task in program code. We show how, given information about the actions possible in the environment, a reward machine can be used to generate tree and rollout policies for a MCTS-based scheduler that allow an agent to predict the actions of other agents based only on the task they are performing. We evaluate our approach in a range of multi-agent environments, and show that I_{RM} out-performs previous intention-aware scheduling approaches in settings where agents are not co-designed.

2 BDI Agents

In this section, we introduce the basic components of a BDI agent, including beliefs, goals, plans and goal-plan trees.

Beliefs and goals. We assume a finite set of propositions P. $S \subseteq \wp(P)$ is a non-empty finite set of environment states (truth assignments to propositions in P). The agent's *beliefs* $B = \{b_1, \ldots, b_n\}$ are a finite set of ground literals defined over P representing its information about the environment. For simplicity, we assume the environment is fully observable, and the agent's beliefs are updated when the state of the environment changes. The agent's *top-level goals* $G = \{g_1, \ldots, g_m\}$ are a finite set of ground literals representing states that the agent wants to bring about. Note that G does not need to be consistent, as conflicting goals may be achieved at different times.

Actions and plans. An agent can perform a set $Act = \{\alpha_1, \ldots, \alpha_k\}$ of primitive *actions* in the environment. The *preconditions* of an action α_i, $\phi = pre(\alpha_i)$, are a set of literals that must be true for the action to be executable, and the *postconditions* of α_i, $\psi = post(\alpha_i)$, are a set of literals 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* if $B \models \phi$. 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. 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.

Goal-plan trees. The relationship between the plans, actions and subgoals that can be used to achieve a goal can be 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 plans that achieve the top-level goal. As only one plan must be executed to achieve the goal, goal nodes can be viewed as or-nodes. The children of a plan node are the action and subgoal nodes corresponding to the steps in the plan body. As all the child nodes of a plan node must be executed to achieve the goal, plan nodes can be viewed as (ordered) and-nodes. Each subgoal node has its associated plans as children. The resulting tree structure represents all possible ways an agent can achieve the top-level goal.

3 Multi-Agent Intention Progression with Reward Machines

In this section, we present our approach to multi-agent intention progression. We assume that we have a BDI agent operating in an environment containing k other agents. Unlike [Dann *et al.*, 2020; Dann *et al.*, 2021] the programs of the other agents are not assumed to be similar to those of the BDI agent; only a high-level declarative specification of the other agents' goals or tasks specified in appropriate formal language, such as LTL or LTL$_f$, is given. The key idea underlying our approach is to use these declarative specifications and the actions possible in the environment to predict the likely behaviour of other agents. To simplify notation, we present our approach for a single other agent; however it is straightforward to extend it to multiple other agents.

Rather than work directly with LTL specifications, we focus on tasks represented by *reward machines*. Reward machines can be computed from task specifications expressed in a range of goal and property specification languages, including LTL and LTL$_f$, in a straightforward way (given the reward for satisfying the specification). In the interests of brevity, we refer the reader to [Camacho *et al.*, 2019] for details of reward machine generation.

A reward machine (RM) [Toro Icarte *et al.*, 2018] is a Mealy machine where states represent abstract 'steps' or 'phases' in a task, and transitions correspond to observations of *high-level events* in the environment indicating that an abstract step/phase in the task has (or has not) been completed. We assume that events are sets of literals over P (we denote the set of all literals over P by $\bar{P}$). Formally, we require that events are generated by a *labelling function* $L : S \times Act \times S \to \wp(\bar{P})$; for example, a labelling could consist of postconditions of actions. An agent may have several goals or tasks, each represented by a reward machine.

Definition 1 *A reward machine is a tuple* $R = (U, u_I, \Sigma, \delta_R, r_R)$ *where:*

- U *is a finite non-empty set of states;*
- u_I *is the initial state;*
- Σ *is a finite set of environment events;*

- $\delta_R : U \times \Sigma \to U$ *is a transition function that, for every state* $u \in U$ *and environment event* $\sigma \in \Sigma$*, gives the state resulting from observing event* σ *in state* u*; and*
- $r_R : U \times \Sigma \to \mathbb{R} \cup \{-\infty\}$ *is a reward function that for every state* $u \in U$ *and event* $\sigma \in \Sigma$ *gives the reward resulting from observing event* σ *in state* u*.*

In line with the approach to *taskable* RL [Illanes *et al.*, 2020], we restrict our attention to *task completion* reward machines which, on producing a non-zero reward, go into a final state from where it is not possible to make a transition to any other state. We denote by F_R the set of 'positive reward' states of R, that is, states u' such that for some u, σ, $\delta_R(u, \sigma) = u'$ and $r_R(u, \sigma) > 0$. For each $u' \in F_R$, let $r_R^-(u') = r_R(u, \sigma)$. Similarly we define the set of 'infinite negative reward states' $N_R = \{u' \mid \exists u \exists \sigma . \delta_R(u, \sigma) = u' \wedge r_R(u, \sigma) = -\infty\}$. N_R states represent violation of invariant properties.

Reward machines were originally proposed as a way of increasing sample efficiency in reinforcement learning, by making the high-level structure of a task available to guide learning. However, in our approach, they are used to formalise the abstract goal-directed behaviour of agents regardless of how the agent is programmed.

We represent the actions the other agent may perform using an *action automaton*, in which the states are states of the environment, and the transitions are the actions that may be performed in a state.

Definition 2 *An action automaton is a tuple* $A = (S, Act, d, \delta_A)$ *where:*

- $S \subseteq \wp(P)$ *is a non-empty finite set of states (truth assignments to propositions in* P*);*
- Act *is a non-empty finite set of actions;*
- $d : S \to \wp(Act) \setminus \{\emptyset\}$ *is a function which assigns to each* $s \in S$ *a non-empty set of actions; and*
- $\delta_A : S \times Act \to S$ *is a partial function that for every* $s \in S$ *and action* $\alpha \in d(s)$ *gives the state resulting from executing* α *in* s*.*

For an agent with tasks specified by reward machines $R_1, \ldots, R_m$ over possibly different alphabets Σ_i corresponding to labelling functions L_i and an action automaton A, we define a *task automaton* characterising the current step of each task and the reward obtained by the agent on performing an action α as:

Definition 3 *A task automaton* $M = A \times R_1 \times \cdots \times R_m$ *is a tuple* $(Q, Q_I, Act \times \Sigma_1 \times \cdots \times \Sigma_m, \delta_M, r_M)$ *where:*

- $Q = S \times U_1 \times \cdots \times U_m$*;*
- $Q_I = S \times \{u_{I_1}\} \times \cdots \times \{u_{I_m}\}$*;*
- $(s', u'_1, \ldots, u'_m) = \delta_M((s, u_1, \ldots, u_m), (\alpha, L_1(s, \alpha, s'), \ldots, L_m(s, \alpha, s'))$ *iff:*
  - $s' = \delta_A(s, \alpha)$ *in the action automaton;*
  - $u'_i = \delta_{R_i}(u_i, L_i(s, \alpha, s'))$ *in each reward machine* R_i*; and*
- $r_M((s, u_1, \ldots, u_m), (\alpha, L_1(s, \alpha, s'), \ldots, L_m(s, \alpha, s'))) =$
  - $\Sigma_{i=1}^m r_{R_i}^-(\delta_{R_i}(u_i, L_i(s, \alpha, s')))$ *iff for all* i*,* $\delta_{R_i}(u_i, L_i(s, \alpha, s')) \in F_{R_i}$*;*
  - $-\infty$ *iff for some* i*,* $\delta_{R_i}(u_i, L_i(s, \alpha, s')) \in N_{R_i}$*;*
  - 0 *otherwise.*

Each state of a task automaton is a tuple of the environment state and the states of the reward machine for each task, and transitions are parallel transitions of the reward machines on receiving the event corresponding to the execution of an action. Note that the rewards are obtained by the agents only when *all* reward machines have reached positive reward states. Intuitively, the definition of r_M says that the agent whose actions we are predicting will continue until all its tasks are achieved (when it gets the sum of the rewards for each task), or it violates an invariant property (when it gets a reward of $-\infty$).

To predict which actions an agent may perform next we use the *tactic set* of the task automaton:

Definition 4 *The* tactic set $T(M, \epsilon)$ *of a task automaton* $M = A \times R_1 \times \cdots \times R_m$ *is the set of all triples of the form* (q, γ, h)*, where* $q \in Q$*,* $\gamma = (\alpha, \sigma_1, \ldots, \sigma_m) \in Act \times \Sigma_1 \times \cdots \times \Sigma_m$*, and* $h = r_{max} \times \epsilon^n$ *is the expected reward of* α *in* q*.* $r_{max} = max(\{r_M(q', \gamma') \mid q' \in Q, \gamma' \in Act \times \Sigma_1 \times \cdots \times \Sigma_m\})$ *is the maximal possible award,* n *is the length of the shortest path in* M *from* q *to a state with* r_{max} *reward where the first action on the path is* α*, and* $\epsilon \in (0, 1)$ *is a discount factor.*

ϵ penalises unnecessary actions and paths with excessive number of steps. The expected reward for an action α in state q depends on the number of actions that must be taken after α to reach a r_{max} state. Each action discounts r_{max} by ϵ, i.e., a path of length n discounts r_{max} by ϵ^n; as a result, actions that are the first step in shorter paths have higher expected rewards. As we will see below, the precise value of ϵ is not important, so long as it is sufficiently close to 1. Giving the agent a reward only when all tasks have been achieved ensures that the tactic set is insensitive to the particular value of the discount factor ϵ.

The computation of the tactic set is shown in Algorithm 1, and is essentially the least fixpoint of the existential preimage function:

$$pre(\gamma, X, \epsilon) = \{(q, \gamma, h \times \epsilon) \mid \exists q' . \delta_M(q, \gamma) = q' \wedge (q', \gamma', h) \in X \wedge \neg \exists h' . (q, \gamma, h') \in X \wedge h' \geq h)\}$$

Starting from a set of (q, γ, r_{max}) triples where $r_{max} = max(\{r_M(q, \gamma) \mid q \in Q, \gamma \in Act \times \Sigma_1 \times \cdots \times \Sigma_m\}$, that is, states where the agent has completed all its tasks (without violating an invariant) and obtained an r_{max} reward, we work backwards. At each iteration, we add to the tactic set triples (q, γ, h), where q is a state in the task automaton from which

Algorithm 1 Computation of the tactic set.

function TACTIC-SET(M, ϵ)
  $T_1 \leftarrow \emptyset$; $T_2 \leftarrow \{(q, \gamma, r_{max}) \mid r_{max} = r_M(q, \gamma)\}$
  while $T_2 \not\subseteq T_1$ **do**
    $T_1 \leftarrow T_1 \cup T_2$; $T_2 \leftarrow \bigcup_{\gamma \in Act \times \Sigma_1 \times \cdots \times \Sigma_m} pre(\gamma, T_1, \epsilon)$
  return T_1

a state q' in the current tactic set is reachable, γ is the action the agent should perform to reach q', and where the h value is maximal. That is, we do not include in the tactic set (q, γ, h) triples, where we have already found a shorter path from q to an r_{max} state which uses a different action.

While the size of the task automaton is exponential in the number of tasks, computation of the tactic-set is polynomial in M.

Theorem 1 *The tactic set can be computed in time polynomial in the size of* $A \times R_1 \times \cdots \times R_m$.

The proof is immediate from Algorithm 1. We note that it is often beneficial to compute the task automaton and the tactic set for only the *current* state of the environment and recompute it when the environment changes (or changes "enough"), as this gives an exponentially more compact representation of the behaviour of the other agent, compared to, e.g., a policy for an RL agent that specifies which action to take in all possible environment states.

The tactic set can be seen as an approximation of the behaviour of a rational agent for a given task, in the sense that it specifies, for each state s and action α, the length of the shortest sequence of actions starting with α to reach the goal. If actions have equal cost, we would expect a rational agent to select actions that reach the goal in fewer steps. For example, an optimal policy for a reinforcement learning agent in an environment specified by A, rewards specified by R, and a sufficiently large discount factor, will assign nonzero probability only to actions with the highest value of h. Similarly, a model-based agent using A^* search with a state space S and a set of actions Act, will execute a plan where, in each state s, an action α with highest h (corresponding to the shortest distance to the goal). For agents that are programmed, such as BDI agents, the situation is less clear cut. However, it is reasonable to assume that a well-written program for a task will tend to execute actions that have the highest value of h in a given state.

4 Intention-Aware Scheduling with RMs

In this section, we show how tactic sets derived from task automata can be used to generate rollouts for an intention-aware MCTS-based scheduler, which we call I_{RM}. I_{RM} is based on the state-of-the-art I_B scheduler [Dann *et al.*, 2021] which uses multiplayer MCTS to decide which intention it should progress next. When performing simulations, I_B assumes that other agents will pursue their goals in the same way as the I_B agent would; that is, external agents are modelled using I_B's own (partially-ordered) GPTs. In contrast, I_{RM} simulates the behaviour of other agents using tactic sets. This allows I_{RM} to anticipate actions by other agents that an I_{RM} agent itself would never take. In the remainder of this section, we briefly outline the modifications to I_B.

Modifications to the Expansion Phase of I_B. In the *expansion* phase of MCTS, a child node is added for each action available at the selected leaf node. In I_B, the set of actions available to the other agent is determined by considering the next steps in the external agents' (inferred) GPTs and their associated preconditions. In I_{RM}, there are two types of node expansion. At nodes where it is the I_{RM} agent's turn to act, expansion proceeds as under I_B. However, at nodes where it is another agent's turn to act, the set of expandable actions is inferred from the tactic set used to model the agent.

Modifications to the Rollout Phase of I_B. During the *rollout* phase of MCTS, I_{RM} simulates the I_{RM} agent's actions in the same way as I_B (via random intention progression). However, it assumes that other agents will attempt to maximise the heuristic value, h, of the tactic set. To account for the fact that maximising h may not be optimal in multi-agent settings, and that external agents may not act wholly rationally, we use a softmax distribution over h to yield a noisy rollout policy. The *temperature parameter*, τ, of the softmax distribution controls the stochasticity of the other agents' rollout behaviour.

5 Evaluation

To evaluate our approach, we extend two popular domains from the reward machine literature, *Office World* [Toro Icarte *et al.*, 2018] and *Craft World* [Andreas *et al.*, 2017], to a multi-agent setting. As in Dann et al. [2020; 2021], we focus on two-agent scenarios and consider cooperative, selfish and adversarial settings.[1]

In Section 5.2, we study a cooperative task in the Office World domain. This experiment highlights the hazards in pairing previous MCTS-based schedulers, which assume an egocentric model of other agents, with agents that are not co-designed. In Section 5.3, we study four competitive tasks in the Craft World domain. In this experiment, we consider a range of external agents, from fully co-designed agents that follow the same GPTs to non-co-designed agents, and study how the degree of design overlap affects I_{RM} and other MCTS-based schedulers.

5.1 Experimental Setup

In the experiments, all MCTS-based schedulers assume an equal goal weighting of 1. Episodes terminated due to failure yield a rollout score of -1. MCTS is configured to maximise the discounted return, with a discount factor of 0.99. Where no intentions are progressable, or simulation suggests that the currently selected plan results in lower reward than another plan, the algorithm attempts to replan by selecting a new plan for the top-level goal in each GPT. Replanning is handled by allowing the agent to select from any plan at the root node of the MCTS search tree. Note that allowing the agent to consider alternatives to its currently intended plan is low-cost: with MCTS, plans that yield larger average simulated returns automatically receive more of the computation budget. When no agent can act, even after replanning, the episode is considered complete.

In both grid world domains considered, the action automaton used by I_{RM} assumes that the other agent can move in the four cardinal directions (unless blocked by a wall) and collect/use items at the current grid square. In configuring

[1] Full source code and demo videos of the agents are available at: https://github.com/mchldann/IRM_IJCAI.

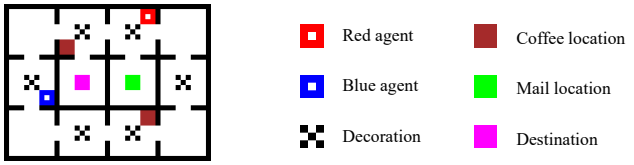

Figure 1: The Office World domain.

the stochasticity of the external agent's rollouts, we considered $\tau \in \{0.005, 0.01, 0.015, 0.02\}$ and selected $\tau = 0.015$.

5.2 Office World Experiments

The Office World domain (Figure 1) is a simulated office environment where agents can collect coffee and mail from designated locations (brown and green squares, respectively), and deliver them to a destination (pink square) without stepping on any decorations (squares with a cross).

The delivery task can can be described by the following LTL$_f$ formula:

$$\begin{aligned}\varphi = &\Box(\neg d) \wedge (\Diamond(m \wedge \Diamond(c \wedge \bigcirc\Diamond(o))) \vee \Diamond(c \wedge \Diamond(m \wedge \bigcirc\Diamond(o))))\\ &\wedge \Box(m \rightarrow \Box(c \rightarrow \bigcirc\Box(o \leftrightarrow \mathit{final})))\\ &\wedge \Box(c \rightarrow \Box(m \rightarrow \bigcirc\Box(o \leftrightarrow \mathit{final})))\end{aligned}$$

where c represents coffee, m the mail, o the office, and d decorations. Note that it is allowable for the agent to pick up one item first, deliver it to the office and then deliver the second item. Moreover, if the agent gets the mail first, then after picking up the coffee the agent can only reach the office at the final time. Similarly, if the agent gets the coffee first, then after picking up the mail the agent can only reach the office at the final time. This ensures that the agent does not get rewarded multiple times. The above LTL$_f$ formula can be translated into the reward machine shown in Figure 2.

We consider a task where two agents (*blue* and *red*) must each deliver both coffee and mail to the destination. Delivering both items is treated as a single goal. At the start of each episode, each agent is spawned randomly in different rooms. Stepping on a decoration (a black cross) leads to termination for the offending agent. Additionally, it is considered dangerous for the agents to occupy the same room, with the task terminating for both agents if this occurs. The first agent to deliver both coffee and mail is deactivated, allowing the other agent to enter the destination room.

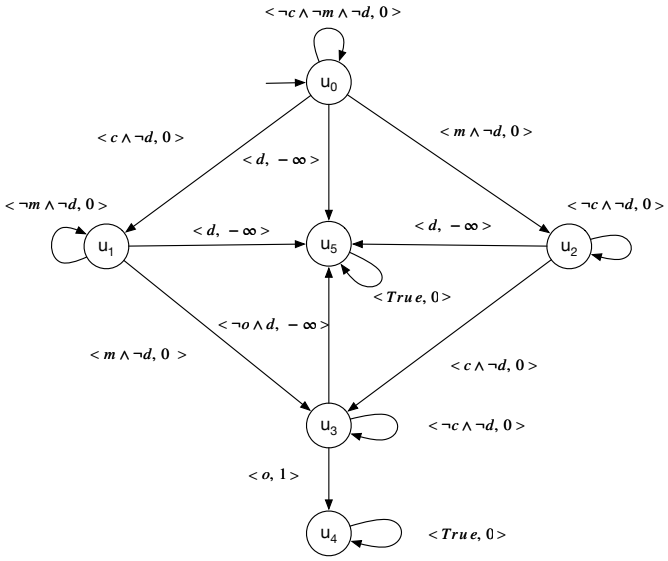

Figure 2: Reward Machine for delivering coffee and mail.

We paired the blue agent with three types of red agent:

- A*: an agent that follows the shortest route to the goal, as calculated by A* on the assumption that the blue agent will never obstruct it.
- $RL_{\tau=0.01}$: a reinforcement learning agent trained on I_{RM}'s reward machine (described below), following a softmax policy over the optimal Q-values with $\tau = 0.01$.
- $RL_{\tau=0.02}$: as above, but configured to behave more erratically with $\tau = 0.02$.

Note that all three red agents effectively follow single-agent policies and ignore the possibility of ending up in the same room as the blue agent.

We compare the performance of the blue I_{RM} agent against two other MCTS-based approaches: the state-of-the-art I_B scheduler [Dann *et al.*, 2021], and the S_P scheduler from [Yao *et al.*, 2016b]. The intention-aware schedulers, I_{RM} and I_B, are configured to behave cooperatively, i.e. to maximise the combined, discounted rewards of the red and blue agents. As S_P is based on single-player MCTS it effectively ignores the red agent and always behaves "selfishly" (i.e., it seeks to maximise its own return).

To generate GPTs for the I_{RM}, I_B and S_P agents, we wrote a GPT generator that constructs multiple plans for delivering the coffee and mail. The generated GPTs contain plans that allow the agents to travel both clockwise or anticlockwise to the key locations, so that the blue I_{RM} and I_B agents can attempt to avoid the red agent. Actions that move the agent to a new room have a precondition that the red agent is not already in that room, making it impossible for the blue agents to instigate a collision. The plans constructed by the generator also avoid stepping on decorations. Since I_B models other agents using its own GPTs, it assumes that *neither* agent will ever step on decorations. To configure I_{RM} similarly, we set its reward machine to pay a penalty of $-\infty$ for stepping on decorations.

The preconditions of the movement actions in the plans to move from one location to another effectively mean that the movement plans for different goals cannot be interleaved, and the steps in a plan must be executed in sequence. As such, the generated plans are totally-ordered, and the I_B scheduler, which supports partially-ordered GPTs, behaves equivalently to the earlier I_A scheduler [Dann *et al.*, 2020] in this domain.

Baseline Compatibility Issues. It is important to note that, strictly speaking, I_B (and I_A) are incompatible with the red agents used in the experiment, as the red agents are not GPT based. Both I_B and I_A assume that it is possible to infer the progression of the other agents' GPTs from their actions, and this is required during node expansion and to determine the starting point of the MCTS simulations. A major advantage of our proposed I_{RM} scheduler is that it does not have

		Red agent		
		A*	$RL_{\tau=0.01}$	$RL_{\tau=0.02}$
Blue agent	S_P	0.40	0.37	0.44
	I_B	0.62	0.64	0.73
	I_{RM}	**0.91**	**0.96**	**0.97**

Table 1: Office World completion rates, averaged over 100 episodes.

this limitation. Nonetheless, to allow comparison against I_B, we configured the GPT generator to construct plans *for every possible agent position in the grid*. This ensures that, however the red agent moves, I_B's model of the red agent always includes applicable plans (allowing it to simulate the red agent's behaviour). However, this results in extremely large GPTs, and for many domains it would be infeasible. Without this, I_B is often unable to find an applicable plan for the red agents, leaving it unable to simulate their behaviour. In this case, I_B's performance reverts to that of S_P, making S_P arguably a more realistic baseline.

Results

The results of the Office World experiments are summarised in Table 1. Since there are only two possible outcomes (either both agents succeed or both agents fail), the results are expressed as success rates.

There is an intuitive explanation as to why I_{RM} significantly outperforms I_B and S_P. As I_B models the red agent via its own GPTs and S_P does not model it at all, neither scheduler sees the possibility of both the red and blue agents being in the same room (since the preconditions of their own plans means this cannot happen). This clearly illustrates the risks in assuming that other agents will behave in the same way as oneself. I_{RM} still *expects* the red agent to try to avoid collisions, since collisions lead to poor rollout scores for both agents, but it sees a non-zero probability of collisions occurring, and hence tries to avoid them.

While I_B and S_P are blind to the risk of collisions, they still see that they cannot pass through the red agent's room. Therefore, they still try to avoid the red agent, albeit less carefully than I_{RM}. However, since I_B is intention-aware and S_P is not (S_P effectively assumes that the red agent will remain stationary), I_B computes better estimates of the red agent's future position. As a result, I_B has fewer collisions than S_P, despite being unaware of all the risks.

5.3 Craft World Experiments

In the Craft World domain (Figure 3), agents must collect resources and use them to craft various items. For example, collected wood can be crafted into a stick at a workbench, then combined with iron at a toolshed to craft an axe. The axe can then be used to mine a gem. The full list of crafting rules is given in Table 2.

We consider environments with two agents, *blue* and *red*, and evaluated the agents in four scenarios listed in Table 3. The shared goal items are required by both agents, while the agent-specific items in Scenarios 2 and 4 are assigned randomly to the agents (one item each). In Scenarios 1 and 2, the blue agents (I_{RM}, I_B and S_P) are configured to behave selfishly, i.e. to maximise their own goal achievement. The selfish (neutral) setting models the case when the red agent is not cooperative (does not get a reward when the blue agent completes a task). In Scenarios 3 and 4, they are configured to behave adversarially, i.e., to maximise $blue\_goals - red\_goals$. Since the resources are limited (there are 3 each of wood, grass and iron), the domain is inherently competitive in both cases. The positions of the resources, the agents and the crafting locations are randomised at the start of each episode.

Figure 3: The Craft World domain.

Goal	Resources	Tools	Location
Axe	Iron, stick		Toolshed
Bed	Grass, plank		Workbench
Bridge	Iron, wood		Factory
Cloth	Grass		Factory
Gem		Axe	
Gold		Bridge	
Plank	Wood		Toolshed
Rope	Grass		Toolshed
Stick	Wood		Workbench

Table 2: Rules for acquiring goal items in Craft World.

As in the OfficeWorld experiments, we wrote a generator to construct GPTs for the blue agents (again accommodating I_B by including every possible location). However, rather than generating an exhaustive list of all possible plans, for each goal item, the generator only considers 2 out of 3 locations for each resource, and only one of the two toolshed locations. This is intended to mimic the situation where a developer has a preference for the ways in which the goals are achieved, and allows us to study how the blue agents' performance is affected by the degree to which the red and blue agents are co-designed. In addition to the A* and RL agents, we consider two additional classes of red agent:

- *Fully co-designed* S_P, I_B and I_{RM} schedulers, which share the same GPTs as the blue agents.
- *Partially co-designed* S_P, I_B and I_{RM} schedulers, which use the same generator as the blue agents, but with a different seed (meaning they may consider different resources / toolshed locations).

Scn.	Objective	Shared goal items	Agent-specific
1	Selfish	Gold, axe, bed	
2	Selfish	Gem, axe, bridge	Rope / Cloth
3	Adversarial	Stick, plank, cloth, rope	
4	Adversarial	Gold, bridge, cloth	Rope / Plank

Table 3: The Craft World scenarios.

		Red agent								
		Fully co-designed			Not co-designed			Partially co-designed		
		S_P	I_B	I_{RM}	A*	$RL_{\tau=0.01}$	$RL_{\tau=0.02}$	S_P	I_B	I_{RM}
Scn. 1	S_P	1.44	1.17	1.19	1.13	1.31	1.69	1.47	1.27	1.27
	I_B	**1.76**	**1.44**	**1.51**	1.33	1.48	1.79	1.71	**1.57**	**1.47**
	I_{RM}	1.71	1.43	1.47	**1.52**	**1.59**	**1.89**	**1.75**	1.53	**1.47**
Scn. 2	S_P	2.78	2.50	2.57	2.61	2.71	3.27	2.96	2.74	2.63
	I_B	**3.17**	**2.85**	**2.85**	2.91	3.00	3.36	3.05	2.87	2.84
	I_{RM}	**3.17**	2.80	2.80	**3.27**	**3.24**	**3.54**	**3.06**	**2.94**	**2.92**
Scn. 3	S_P	0.01	-0.68	-0.25	-0.64	-0.50	0.19	-0.04	-0.22	-0.23
	I_B	**0.60**	**0.00**	**0.33**	-0.63	-0.31	0.50	0.11	0.00	-0.07
	I_{RM}	0.32	-0.38	0.02	**-0.15**	**0.03**	**0.81**	**0.32**	**0.10**	**0.01**
Scn. 4	S_P	0.05	-0.62	-0.59	-0.71	-0.52	0.18	-0.01	-0.32	-0.57
	I_B	**0.60**	**-0.05**	**0.10**	-0.51	-0.32	0.55	0.42	-0.04	-0.32
	I_{RM}	0.48	-0.06	0.02	**0.17**	**0.24**	**0.93**	**0.68**	**0.32**	**0.01**

Table 4: Craft World results, averaged over 100 episodes. The best results are highlighted in bold.

Results

The results for the Craft World experiments are shown in Table 4. In the selfish scenarios (1 and 2), the reported score is the mean number of goal items crafted by the blue agent. In the adversarial scenarios (3 and 4), the reported score is the mean of $blue\_goals - red\_goals$.

The experiments where the red and blue agents are fully co-designed represent the best case for I_B, since I_B knows which toolshed and resources the other agent will use, while I_{RM} does not. As expected, I_B performed best here, top scoring in 11 out of 12 cases and drawing with I_{RM} in the other. While I_{RM} underperformed I_B in the fully co-designed setting, it easily beat S_P in all cases and only marginally underperformed I_B in Scenarios 1 and 2.

At the other end of the spectrum, I_{RM} outperformed I_B in all cases where they were paired with non-co-designed agents. I_B is "surprised" when the red agent collects a resource or uses a toolshed that is not available to itself. While I_{RM} is also unable to use these resources, it is able to anticipate that the red agent may use them. Intuitively, this understanding ought to be especially important in the adversarial scenarios, and I_{RM} did in fact win by large margins there.

The partially co-designed case lies somewhere in the middle; sometimes I_B's assumptions about the red agent's GPTs are correct, allowing it exploit knowledge that is unavailable to I_{RM}, while at other times it makes incorrect assumptions and is unable to predict the red agent's behaviour. While this situation does not favour either agent, I_{RM} was clearly the stronger performer, winning in 10 out of 12 cases and drawing with I_B in another.

Overall, while I_B outperforms I_{RM} when paired with fully co-designed agents, I_{RM} performs better with agents that are not or only partially co-designed. Moreover, as we noted earlier, S_P is arguably a fairer baseline in practical settings (where the GPTs cannot easily be expanded to accommodate I_B). Since I_{RM} beat S_P in all cases, I_{RM}'s advantage over I_B is likely to be even greater in practice.

6 Discussion and Conclusion

In this paper we introduced I_{RM}, a new MCTS-based intention-aware scheduler for BDI agents that uses reward machines derived from declarative task specifications to predict the behaviour of other agents. Unlike previous approaches, I_{RM} does not assume that the behaviour of other agents will be similar to its own. In our experiments, this allowed I_{RM} to act more safely in the Office World task, while in Craft World it outperformed the state-of-the-art I_B scheduler for all paired agents except those based on fully co-designed GPTs.

In addition to the work on intention scheduling mentioned in the introduction, our approach also has some similarities to agents that combine MCTS with reinforcement learning, especially those for multiplayer games such as Go [Gelly and Silver, 2008; Silver *et al.*, 2016]. However I_{RM} models the tasks of the other agents in its environment declaratively, which makes it easy to adapt an I_{RM}-based BDI agent when the goals of the other agents change. The work of Illanes *et al.*'s [2020] on combining hierarchical RL with symbolic planning is also somewhat related to our approach, as its notion of "final-state goal tasks" is reminiscent of the way in which we combine multiple reward machines, where the agent only receives a payoff once all tasks are complete. However, their work focuses on single agent tasks, and the agent's low-level behaviour is unconstrained.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (61906169).

References

[Andreas *et al.*, 2017] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, pages 166–175. PMLR, 2017.

[Broekens *et al.*, 2010] Joost Broekens, Maaike Harbers, Koen V. Hindriks, Karel van den Bosch, Catholijn M.

Jonker, and John-Jules Ch. Meyer. Do you get it? user-evaluated explainable BDI agents. In *Proceedings of the 8th German Conference on Multiagent System Technologies (MATES 2010)*, volume 6251 of *LNCS*, pages 28–39. Springer, 2010.

[Camacho *et al.*, 2019] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 6065–6073. IJCAI, 2019.

[Dann *et al.*, 2020] Michael Dann, John Thangarajah, Yuan Yao, and Brian Logan. Intention-aware multiagent scheduling. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020)*, pages 285–293. IFAAMAS, 2020.

[Dann *et al.*, 2021] Michael Dann, Yuan Yao, Brian Logan, and John Thangarajah. Multi-agent intention progression with black box agents. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*. IJCAI, 2021.

[de Silva *et al.*, 2020] Lavindra de Silva, Felipe Meneguzzi, and Brian Logan. BDI agent architectures: A survey. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*. IJCAI, 2020.

[Gelly and Silver, 2008] Sylvain Gelly and David Silver. Achieving master level play in 9 x 9 computer go. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 1537–1540. AAAI Press, 2008.

[Illanes *et al.*, 2020] León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, pages 540–550. AAAI Press, 2020.

[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. IFAAMAS, 2017.

[Norling and Ritter, 2004] Emma Norling and Frank E. Ritter. Towards supporting psychologically plausible variability in agent-based human modelling. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, pages 758–765. IEEE Computer Society, 2004.

[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. Morgan Kaufmann, 1992.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[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.*, 2003] John Thangarajah, Lin Padgham, and Michael Winikoff. Detecting & avoiding interference between goals in intelligent agents. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 721–726. Morgan Kaufmann, 2003.

[Toro Icarte *et al.*, 2018] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, pages 2107–2116. PMLR, 2018.

[Waters *et al.*, 2014] Max Waters, Lin Padgham, and Sebastian Sardiña. Evaluating coverage based intention selection. In *Proceedings of the 13th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2014)*, pages 957–964. IFAAMAS, 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 (AAMAS 2016)*, pages 1227–1236. IFAAMAS, 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)*, pages 1133–1134. ECCAI, IOS Press, 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)*, volume 10093 of *LNCS*, pages 176–191. Springer, 2016.

[Yao *et al.*, 2016b] 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-16)*, pages 2558–2565. AAAI Press, 2016.