Intention Progression using Quantitative Summary Information

Yuan Yao
Zhejiang University of Technology
yaoyuan@zjut.edu.cn

Natasha Alechina
Utrecht University
n.a.alechina@uu.nl

Brian Logan
Utrecht University
b.s.logan@uu.nl

John Thangarajah
RMIT University
john.thangarajah@rmit.edu.au

ABSTRACT

A key problem for Belief-Desire-Intention (BDI) agents is *intention progression*, i.e., which plans should be selected and how the execution of these plans should be interleaved so as to achieve the agent's goals. Monte-Carlo Tree Search (MCTS) has been shown to be a promising approach to the intention progression problem, out-performing other approaches in the literature. However, MCTS relies on runtime simulation of possible interleavings of the plans in each intention, which may be computationally costly. In this paper, we introduce the notion of *quantitative summary information* which can be used to estimate the likelihood of conflicts between an agent's intentions. We show how offline simulation can be used to precompute quantitative summary information prior to execution of the agent's program, and how the precomputed summary information can be used at runtime to guide the expansion of the MCTS search tree and avoid unnecessary runtime simulation. We compare the performance of our approach with standard MCTS in a range of scenarios of increasing difficulty. The results suggest our approach can significantly improve the efficiency of MCTS in terms of the number of runtime simulations performed.

KEYWORDS

Intention progression; BDI agents; Qualitative summary information; Monte-Carlo Tree Search

ACM Reference Format:
Yuan Yao, Natasha Alechina, Brian Logan, and John Thangarajah. 2021. Intention Progression using Quantitative Summary Information. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 9 pages.

1 INTRODUCTION

In Belief-Desire-Intention-based (BDI) agents [14] the behaviour of an agent is specified in terms of beliefs, goals, and plans. *Beliefs* represent the agent's information about the environment (and itself), *goals* represent desired states of the environment the agent is trying to bring about, and *plans* are the means by which the agent can achieve its goals. Plans consist of primitive actions that directly change the state of the environment, and subgoals which are in turn achieved by subplans. When the agent commits to a particular plan to achieve a (top-level) goal, an *intention* is formed. At each deliberation cycle, the agent chooses which of its multiple intentions it should progress (i.e., intention selection) and, if the next step within the selected intention is a subgoal, chooses the best plan to achieve it (i.e., plan selection). These two choices together form the *intention progression problem* [13].

An agent typically pursues multiple goals in parallel. In many BDI agent programming languages and platforms, by default, the intentions for each of the agent's top-level goals are progressed in round robin fashion, i.e., at each deliberation cycle, the next step of the next intention in sequence is executed [2, 32]. The resulting interleaving of steps in plans in different intentions may result in conflicts, i.e., the execution of a step in one plan may make the execution of a step in another concurrently executing plan impossible. Most languages allow a developer to over-ride the default scheduling behaviour to avoid conflicts, e.g., by over-riding a method or by writing a meta-plan. However, it is difficult for the developer to anticipate all possible conflicts in advance, and it would be desirable for the agent to solve the intention progression problem itself, i.e., which plans should be selected and how the execution of these plans should be interleaved so as to achieve the agent's goals without conflicts.

A number of approaches to various aspects of the intention progression problem have been proposed in the literature, including, *summary-information-based* (SI) [21, 22], *coverage-based* (CB) [29, 30] and *Monte-Carlo Tree Search-based* (MCTS) [35–37] approaches. In [37] Yao et al. showed that the MCTS-based approach *SA* out-performed round robin, first in first out, SI, and CB intention progression in static and dynamic environments, both in synthetic domains and a more realistic scenario based on the Miconic-10 elevator domain [12]. However, *SA* relies on runtime pseudorandom simulations of different interleavings of the plans in each intention to determine which intention to progress at the current deliberation cycle. Random simulation has the advantage that all possible interleavings have a probability of being explored. However the random nature of the simulations often results in the generation of many "obviously bad" interleavings, which is computationally costly. If the computational budget is (too) small, the performance of MCTS can be unstable in problems with a large search space, such as intention progression [33].

In this paper, we show how the computational overhead of MCTS-based scheduling approaches can be reduced using summary information. Summary information was introduced in [22] as a way of characterising the definite and potential pre- and post-conditions of different ways of achieving a goal. Using this information, a scheduler could decide to, e.g., delay progressing an intention, if doing so would necessarily result in a conflict with another intention. The summary information in [21–23, 37] is represented as boolean combinations of definite and potential pre- and postconditions, and

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

is essentially qualitative, that is, it does not consider the likelihood that a precondition will be required or a postcondition will be established. As a result, it becomes less informative as the agent's program grows in complexity — with more ways of achieving a goal, very few pre- and postconditions are definite, and many pre- and postconditions become potential. Moreover, the size of the summary information itself grows exponentially with the complexity of the agent's program. (Computing it also takes exponential time, but this is done offline.)

We introduce the notion of quantitative summary information which can be used to estimate the likelihood of conflicts between an agent's intentions. We show how offline simulation can be used to precompute quantitative summary information prior to execution of the agent's program, and how the precomputed summary information can be used at runtime to guide the expansion of the MCTS search tree and avoid unnecessary runtime simulation. We compare the performance of our approach with standard MCTS in a range of scenarios of increasing difficulty. The results suggest our approach can significantly improve the efficiency of MCTS in terms of the number of runtime simulations performed.

2 PRELIMINARIES

In this section, we briefly recall the definition of a goal-plan tree that forms the foundation of our approach, and formally define the intention progression problem.

2.1 Goal-Plan Trees

We use the notion of *goal-plan trees* [21, 22, 34] to represent the relations between goals, plans and actions, and to reason about the interactions between intentions. The root of a GPT is a goal node g, its children are plan nodes that can be used to achieve g. Each plan node contains a sequence of action nodes and (sub)goal nodes. The subgoals have their associated plans as their children, giving rise to a tree structure representing all possible ways an agent can achieve the top-level goal g.

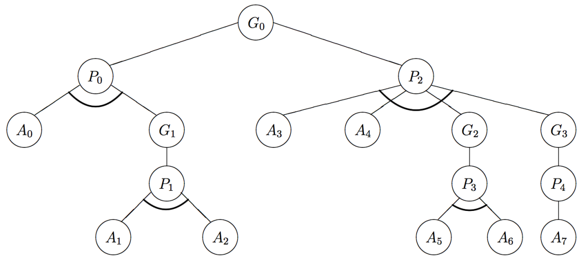

Figure 1: Example goal-plan tree.

Figure 1 shows a simple goal-plan tree. The top level goal, G_0 can be achieved by two plans, P_0 and P_2, and can be seen as an "OR" node, in that the agent has a choice of which plan to use. The plan P_0 consists of a single action, A_0, and a subgoal G_1. Plan nodes can be seen as "AND" nodes, in that, if P_0 is selected, action A_0 must be executed and G_1 must be achieved.

To allow reasoning about interactions between intentions, a goal plan tree records information about the conditions necessary to achieve a (sub)goal or successfully execute a plan or an action in the form of pre- and post-conditions associated with goal, plan and action nodes. *Preconditions* are conditions that must be true in order to execute a plan or an action. *Postconditions* are conditions that are made true by executing a plan or an action or by achieving a goal (in addition to achieving the goal itself). For simplicity, in what follows, we assume that pre- and postconditions are sets of literals. For example, the action A_3 in Figure 1 may have p_0 as precondition and $\{p_1, p_2\}$ as postcondition, A_4 may have p_1 as precondition and p_3 as postcondition, and action A_5 in plan P_3 may have $\{p_2, p_3\}$ as precondition. That is, preceding steps establish preconditions for later steps.

2.2 Intention Progression Problem

The *intentions* of an agent at each deliberation cycle are represented by a pair $I = (T, S)$ where $T = \{t_1, \ldots, t_n\}$ is a set of goal-plan trees and $S = \{s_1, \ldots, s_n\}$ is a set of *pointers to the current step* of each t_i. Each s_i is initially set to the root goal of t_i, g_i. We define $next(s_i)$ as the successor of s_i in a pre-order traversal of t_i (i.e., if s_i is the last step in a plan π, then $next(s_i)$ is the next step in the parent plan of π), and $next_a(s_i)$ as the first action step of t_i following s_i. If $next(s_i)$ is an action, $next_a(s_i) = next(s_i)$. If s_i is a (top-level) goal or $next(s_i)$ is a subgoal, determining the next action step involves choosing a plan π for the (sub-)goal and returning the first action step in π (if the first step in π is again a subgoal, then we also need to choose a plan to achieve it and so on). A current step s_i is *progressable* if there exists an action $a = next_a(s_i)$ whose precondition holds.

The *intention progression problem* (IPP) [13] is that of choosing a current step $s_i \in S$ to progress (i.e., advance to $next_a(s_i)$) at each deliberation cycle so as to maximise the agent's utility. There are many ways in which utility may be defined, e.g., taking into account the importance or priority of goals, the deadlines by which goals should be achieved, the 'fairness' or order in which goals are achieved, the costs or preferences of actions, etc. For concreteness, in what follows we assume that the agent's utility is maximised by achieving the largest number of goals.

3 QUANTITATIVE SUMMARY INFORMATION

In this section, we introduce the notion of quantitative summary information and explain how it can be computed.

3.1 Fragile and Establishing Ratios

We define quantitative summary information in terms of possible executions of the plans an agent may use to achieve its goals. A *trace* is a sequence of primitive actions corresponding to a possible execution of a plan for a goal. Traces are generated by expanding the subgoals in the plan and the plans for those subgoals recursively. The set of execution traces, *traces*(π), induced by a plan π in a goal-plan tree is given by:

$$\begin{aligned} traces(\pi) &= \{a \mid a = expand(e_1), \ldots, expand(e_k)\} \\ &\qquad \text{where } \pi = e_1, \ldots, e_k \text{ and} \\ expand(a) &= a \\ expand(g) &\in traces(\pi') \text{ where } \pi' \in plans(g) \end{aligned}$$

where a is a primitive action, g is a subgoal and $plans(g)$ is the set of plans available to achieve g. A trace $\sigma = a_1, \ldots, a_n$ is *coherent* if, for every pair of steps a_i, a_k such that $i < k$, a_i has a postcondition $\sim l$, a_k has a precondition l, there exists a_j with $i < j < k$ such that a_j has postcondition l. We stipulate that each trace $\sigma \in traces(\pi)$ is coherent.

We characterise traces in terms of the fragility and establishment of literals. We denote by $L(T)$ the set of literals l such that l appears in the postcondition of a goal, plan or action in T, and $\sim l$ appears in the precondition of a plan or action in T. A literal $l \in L(T)$ is *fragile* on an interval $[i, j]$ of a trace $a_1, \ldots, a_n$, iff l is a postcondition of a step a_i, a precondition of a step a_j, and no a_k with $i < k < j$ has l as a postcondition. Intuitively, a literal l is fragile in an interval where it is established but not yet used, and may be clobbered by another interleaved intention that establishes $\sim l$. A special case is when l is true at the beginning of the plan (e.g., l is a precondition of the plan which is true in the environment) and is used for the first time by some a_j, then l is fragile during the interval $[0, j]$. To characterise fragility of a literal l on a trace, we only consider maximal intervals where l is fragile, that is, if l is established by step i and then used by steps j and j', where $j < j'$, without being re-established before j', we only consider the interval $[i, j']$. The notion of an interval where l is fragile is similar to the notion of *preparatory effects* (p-effects) defined in [22]. However we extend the notion of p-effect in [22] to include the establishment of the precondition of an action by a previous action in the same plan, and to include 'unestablished preconditions', i.e., dependencies on prior execution or the environment.

The length of an interval $[i, j]$ is $j - i + 1$. We define the *fragile ratio of l for a plan π, $fr(l, \pi)$*, as the ratio of the sum of lengths of the intervals where l is fragile on all traces of π to the total length of all traces of π,

$$fr(l, \pi) = \frac{\Sigma_{\sigma \in traces(\pi), [i,j] \in F(l,\sigma)} (j - i + 1)}{\Sigma_{\sigma \in traces(\pi)} length(\sigma)}$$

where for $\sigma = a_1, \ldots, a_n$, $F(l, \sigma) = \{[i, j] \mid l \in post(a_i) \land l \in pre(a_j) \land \neg\exists k(i < k < j \land l \in post(a_k)) \land \neg\exists j'(i < j < j' \land l \in pre(a_{j'}) \land \neg\exists k'(i < k' < j' \land l \in post(a_{k'})))\}$. The fragile ratio can be seen as a measure of a plan's robustness: the larger the fragile ratio for l, the more likely interleaving steps from plans in other intentions will destroy (clobber) a fragile precondition l.

We define the *fragile ratio of l for a goal g, $fr(l, g)$* in terms of the fragile ratios of the plans for g,

$$fr(l, g) = \frac{\Sigma_{\pi \in plans(g), \sigma \in traces(\pi), [i,j] \in F(l,\sigma)} (j - i + 1)}{\Sigma_{\pi \in plans(g), \sigma \in traces(\pi)} length(\sigma)}$$

We also define a complementary notion of an establishing ratio. Given a trace $a_1, \ldots, a_n$, a step a_k, $1 \le k \le n$ is *establishing* wrt to a literal $l \in L(T)$ iff l is a postcondition of a_k. For a plan π and a literal l, the *establishing ratio of l on π, $er(l, \pi)$*, is the ratio of the number of steps establishing l on all traces of π, to the total length of all traces of π:

$$er(l, \pi) = \frac{\Sigma_{\sigma \in traces(\pi)} E(l, \sigma)}{\Sigma_{\sigma \in traces(\pi)} length(\sigma)}$$

where $E(l, \sigma)$ is the number of steps establishing l on σ. The establishing ratio can be seen as a measure of a plan's potential to conflict with plans in other intentions: the larger the establishing ratio for a literal l, the more likely interleaving steps from this plan will destroy a fragile precondition $\sim l$ in another intention.

As for fragility, we define the *establishing ratio of l for a goal g, $er(l, g)$* in terms of the establishing ratios of the plans for g,

$$er(l, g) = \frac{\Sigma_{\pi \in plans(g), \sigma \in traces(\pi)} E(l, \sigma)}{\Sigma_{\pi \in plans(g), \sigma \in traces(\pi)} length(\sigma)}$$

3.2 Complexity

As with qualitative summary information, we wish to compute and store quantitative summary information for each goal and plan node in a goal-plan tree so that it can be used at runtime to guide the scheduling of intentions. However, as we show next, computing the fragile and establishing ratios for plans (and hence for goals) is computationally costly.

Proposition 3.1. *Computing $fr(l, g)$ can be done using space polynomial in the GPT representation of $plans(g)$.*

Proof. To show that $fr(l, g)$ can be computed in PSPACE in the size of $plans(g)$, observe that each $\sigma \in traces(\pi)$ for $\pi \in plans(g)$ is linear in the size of π (since it is a concatenation of a subset of branches in π) hence it is linear in the size of $plans(g)$. The number of traces in $traces(\pi)$ in exponential in the size of π, however for each σ we can compute $\Sigma_{[i,j] \in F(l,\sigma)} (j - i + 1)$ one at a time using polynomial amount of space, and keep a running proportion of total length of fragile intervals to the total length of traces explored so far. Note that the values for the numerator and denominator (sum of lengths of traces) may be exponential in the size of $plans(g)$, but they can be represented in binary using only polynomial amount of space. We can keep track of where we are in $plans(g)$ by placing a pointer on a node in a goal-plan tree, again using only polynomial space. □

In the next theorem, we show that the problem of computing $fr(l, g)$ is ♯P-hard. The complexity class ♯P contains function form problems that involve counting solutions to NP problems [24]. Clearly a solution to the problem of counting satisfying assignments is also a solution to the boolean satisfiability problem, so it is at least as hard as an NP-complete problem. This means that it is not possible to compute $fr(l, g)$ in polynomial time in the size of the GPT for g, unless FP=♯P, where FP is the set of function form problems solvable in polynomial time. FP=♯P would entail P=NP, which is unlikely.

Theorem 3.2. *The problem of computing $fr(l, g)$ is ♯P-hard.*

Proof. We reduce the ♯P-complete problem of counting the number of satisfying assignments for a formula in 3CNF to computing $fr(l, g)$. We build on the proof sketch in [20], which shows that the problem of whether there exists an execution of a given GPT is NP-complete by constructing a GPT corresponding to a 3CNF formula and showing that the formula is satisfiable if and only if there is an execution of (coherent trace of a plan in) a corresponding GPT. As in [20], we construct a GPT corresponding to a 3CNF formula ϕ over n variables, that has two plans. One plan is essentially the same as the encoding of ϕ in [20], however each execution of it ends with an action that requires a propositional variable p that does not occur in ϕ. This means that p is fragile in the entire execution corresponding to a satisfying assignment for ϕ. The second plan makes sure that the GPT has an execution corresponding to every one of 2^n possible assignments to the variables in ϕ. The second plan has traces corresponding to assignments satisfying $\neg\phi$. The variable p is not fragile on any of these traces. All executions have the same length. Then ϕ has $k/2^n$ satisfying assignments iff $fr(p, g) = k/2^n$.

The top level goal g in the GPT has two plans, π_ϕ and $\pi_{\neg\phi}$. First we construct π_ϕ. Consider a 3CNF formula ϕ with m clauses. Each clause c_i, $1 \leq i \leq m$, has three literals l_{i1}, l_{i2}, l_{i3}. In a satisfying assignment for ϕ, at least one l_{ij} for each c_i is made true. Following [20], we model this by constructing π_ϕ which is a sequence of subgoals, each corresponding to a clause c_i in ϕ. Each subgoal c_i has three plans, each containing a single action l_{ij} intuitively corresponding to setting the literal l_{ij} to true. In order to make sure that only one of two literals q and $\neg q$ can be set to true in an execution of π_ϕ, each action l_{ij} requires l_{ij} as a precondition, and also has it as a postcondition. So if we execute plan q for subgoal c_i, we cannot execute plan $\neg q$ for a subgoal c_j occurring later in π_ϕ. The last action in plans for subgoal c_m (the last subgoal in π_ϕ) ends in an action p that requires p. Since no action establishes p earlier in π_ϕ, p is fragile for the duration of any trace of π_ϕ. It is also clear that any trace in $traces(\pi_\phi)$ corresponds to a satisfying assignment of ϕ (recall that we require traces to be coherent). The length of all traces of π_ϕ is the same, $m + 1$ (where we execute m actions, one for each clause to set a literal to true, and one for p at the end).

For the second plan $\pi_{\neg\phi}$ we first compute a 3DNF representation of $\neg\phi$: it is a disjunction of m conjuncts $d_1 \vee \ldots \vee d_m$, where d_i is $(\neg l_{i1} \wedge \neg l_{i2} \wedge \neg l_{i3})$. In order to make $\neg\phi$ true, it is sufficient to make one disjunct d_i true, and this means making all three literals in it true. So $\pi_{\neg\phi}$ consists of a single subgoal $s_{\neg\phi}$ with m plans, and each plan for d_i is a sequence of actions corresponding to $\neg l_{i1}, \neg l_{i2}, \neg l_{i3}$, followed by the same sequence of length $m - 2$ trivial steps with no preconditions (to make traces of $\pi_{\neg\phi}$ to be of the same length as the traces of π_ϕ). The variable p is not used in any subplans of $\pi_{\neg\phi}$ and is not fragile on any intervals in those traces. Clearly $plans(g)$ have in total 2^n traces, all of length $m + 1$, and all the ones where p is fragile for all $m + 1$ steps correspond to the satisfying assignments of ϕ. So the ϕ has $k/2^n$ satisfying assignments if, and only if, $fr(p, g) = (m + 1)k/(m + 1)2^n = k/2^n$. □

Proposition 3.3. *The problem of computing $er(l, g)$ is in PSPACE and ♯P-hard.*

The proof is similar to that for Proposition 3.1 and Theorem 3.2 and is omitted due to lack of space.

3.3 Estimating Fragile and Establishing Ratios

The high computational cost makes computation of precise quantitative summary information difficult, even in an offline setting. We therefore adopt an approach based on sampling to estimate the quantitative summary information for each goal and plan in an agent's goal-plan trees. For each plan π in a goal-plan tree $t_i \in T$, we generate v traces using pseudorandom simulation. As the simulations are performed offline, without knowledge of the agent's environment at runtime, we assume that all consistent environments are possible. That is, when performing the simulations, we assume that any unestablished preconditions in a trace hold, e.g., they are established by a parent plan of π or are true in the environment. As we require traces to be coherent, we are guaranteed that each trace is executable in some (consistent) environment.

Given a set of traces $R(\pi)$ produced by pseudorandom simulation of π, for each $\sigma \in R(\pi)$ and literal $l \in L(T)$, we compute the sum of the lengths of the intervals where l is fragile, the number of times l is established and the length of σ. We can then estimate the fragile ratio of l for π, $fr^*(l, \pi)$, as

$$fr^*(l, \pi) = \frac{\Sigma_{\sigma \in R(\pi), [i,j] \in F(l,\sigma)} (j - i + 1)}{\Sigma_{\sigma \in R(\pi)} length(\sigma)}$$

Similarly, we can estimate the establishing ratio of l on π, $er^*(l, \pi)$ as

$$er^*(l, \pi) = \frac{\Sigma_{\sigma \in R(\pi)} E(l, \sigma)}{\Sigma_{\sigma \in R(\pi)} length(\sigma)}$$

The estimates of the fragile and establishing ratios for a goal g, $fr^*(l, g)$ and $er^*(l, g)$ are computed from the sets of traces $R(\pi)$ for each $\pi \in plans(g)$ as follows

$$fr^*(l, g) = \frac{\Sigma_{\pi \in plans(g), \sigma \in R(\pi), [i,j] \in F(l,\sigma)} (j - i + 1)}{\Sigma_{\pi \in plans(g), \sigma \in R(\pi)} length(\sigma)}$$

$$er^*(l, g) = \frac{\Sigma_{\pi \in plans(g), \sigma \in R(\pi)} E(l, \sigma)}{\Sigma_{\pi \in plans(g), \sigma \in R(\pi)} length(\sigma)}$$

Note that the offline computation of fr^* and er^* need only be performed once: it is essentially a static analysis of the agent program and forms part of the development phase rather than the execution of the agent.

4 THE S_Q SCHEDULER

In this section, we present S_Q, an MCTS-based solver for intention progression problems that uses quantitative summary information to guide the expansion of the search tree and avoid unnecessary runtime simulation. To choose which intention to progress, S_Q iteratively builds a search tree. Edges in the tree represent the choice of a $next_a(s_i)$ for one of the agent's intentions. Nodes represent the state of the agent following the execution of the corresponding action, that is, the agent's beliefs updated with the postconditions of the action, and the updated set of current step pointers. In addition, each node also contains the current value of the node and a record of the number of times it has been visited in the search (see below).

Algorithm 1 Return the action to be executed at the current cycle

```
1: function $S_Q(B, S, \alpha, \beta, \gamma, \delta)$
2:     $n_0 \leftarrow node_0(B, S)$
3:     for $i \leftarrow 1, \alpha$ do
4:         $n_e \leftarrow$ MAX-UCT-LEAF-NODE$(n_0)$
5:         $children(n_e) \leftarrow$ EXPAND$(n_e)$
6:         $n_s \leftarrow$ RANDOM-CHILD$(children(n_e))$
7:         $v(n_s) \leftarrow$ VALUE$(n_s, \beta, \gamma, \delta)$
8:         BACKUP$(n_s, v(n_s))$
9:     return MAX-CHILD$(n_0)$
```

S_Q is shown in Algorithm 1 and takes six parameters as input: the agent's current beliefs, B, the set of pointers to the current step in each of the agent's intentions, S, the number of nodes in the MCTS search tree to be expanded at this cycle, α, the number of runtime simulations that may be performed when a node is expanded, β, and two threshold values γ and δ. As with MCTS and *SA* [35], each iteration of the main loop consists of four main phases: selection, expansion, simulation and back-propagation. However the introduction of qualitative summary information requires significant modifications to the simulation phase which are described in detail below.

In the *selection* phase, a leaf node, n_e is selected for expansion (line 4). A node may be expanded if it represents a non-terminal state (i.e., $\exists s_i \in S(n_e)$ such that s_i is progressable). n_e is selected using a modified version of Upper Confidence bounds applied to Trees (UCT) [16], which models the choice of node as a k-armed bandit problem. Starting from the root node, we recursively follow child nodes with highest UCT value until a leaf node is reached.

In the *expansion* phase, n_e is expanded by adding a child node for each $next_a(s_i)$ progressable in state $s(n_e)$ representing the agent's beliefs resulting from executing $next_a(s_i)$ and the updated current step pointer $s'_i = next_a(s_i)$ (line 5). That is, each child node corresponds to a different choice of which intention to progress at this cycle, and, if $next(s_i)$ is a subgoal, how to progress it. One of the newly created child nodes, n_s, is then selected at random for possible simulation (line 6).

In the *simulation* phase, the *value* of n_s is estimated. The function VALUE (see Algorithm 2) takes as argument n_s, the number of simulations we may perform, β, and the upper and lower bounds, $\gamma, \delta, \delta < \gamma$, on the confidence interval within which runtime simulation will be performed. We first compute quantitative summary information for the agent's intentions (line 3). Using the quantitative summary information, we then compute the probability, p, of achieving at least m goals from the state represented by n_s (line 5). If p is less than or equal to δ, we assume achieving m goals is unlikely, and decrement m (line 7). If the probability of achieving m goals is greater than or equal to γ, we return m as the value of n_s (line 9). That is, if the probability of achieving m goals is sufficiently low or sufficiently high, we "trust" the estimate based on qualitative summary information and do not perform any runtime simulations. Otherwise (i.e., $\delta < p < \gamma$ and we are uncertain whether m goals can really be achieved from n_s) we perform β pseudorandom simulations from n_s as in standard MCTS. Starting in the environment represented by n_s, a $next_a(s_i)$ that is executable in $s(n_s)$ is randomly selected and executed, and the environment and the current step of the selected goal-plan tree updated with $s'_i = next_a(s_i)$. This process is repeated until a terminal state is reached in which no next steps can be executed or all top-level goals are achieved. The value of the simulation is taken to be the number of top-level goals achieved in the terminal state. The maximum number of goals achieved in any of the β simulations is returned as the value of n_s (lines 10–13).

Algorithm 2 Return the value of node n_s

```
1: function VALUE($n_s$, $\beta$, $\gamma$, $\delta$)
2:     $m \leftarrow length(intentions(n_s))$
3:     $Q \leftarrow$ QSI$(intentions(n_s))$
4:     while $m \geq 2$ do
5:         $p \leftarrow$ GOALS-ACHIEVED$(intentions(n_s), Q, m)$
6:         if $p \leq \delta$ then
7:             $m \leftarrow m - 1$
8:         else if $p \geq \gamma$ then
9:             return $m$
10:        else
11:            $m \leftarrow 0$
12:            for $i \leftarrow 1, \beta$ do
13:                $m \leftarrow \max(m,$ SIMULATE$(n_s))$
14:            return $m$
```

To compute the fragile and establishing ratios for each of the agent's intentions ι at n_s, the function QSI (see Algorithm 3) processes each unexecuted step in each partially executed plan in ι, accumulating an estimate of the total number of fragile and establishing steps, *fs*, *es*, for each literal l, and the average length of an execution of ι, n. If a step $\iota(j)$ is a subgoal, we use the $fr^*(l, \iota(j))$ and $er^*(l, \iota(j))$ ratios multiplied by the average length of executions of plans for $\iota(j)$, $\iota(j)_{len}$, (computed offline) to estimate the number of fragile and establishing steps for l required to achieve $\iota(j)$, and add these and $\iota(j)_{len}$ to *fs*, *es* and n respectively. For actions, we increment *es* if l is a postcondition of the action, keep track of the number of action steps where l is fragile, r, and increment n. When all steps have been processed, we divide *fs* and *es* by n to estimate *fr* and *er* for l for ι. Note that computing *fs*, *es* and n involves only arithmetic, and so is much less costly than runtime simulation.

The function GOALS-ACHIEVED (see Algorithm 4) uses the quantitative summary information for each intention to calculate the maximum probability of the execution of a subset of m intentions being conflict free, i.e., of achieving m goals. For each pair of intentions, ι_i, ι_j, and literal $l \in L(T)$, we compute the probability that a step in ι_i that is fragile on l will be clobbered by an establishing step in ι_j (ι_j being clobbered by ι_i is symmetric). The number of fragile steps in ι_i, fs_i, is given by the fragile ratio for ι_i, $Q_{fr}(l, \iota_i)$, times

Algorithm 3 Return the QSI for the agent's current intentions

```
1: function QSI($I$)
2:     $Q \leftarrow \emptyset$
3:     for $\iota \in I$ do
4:         for $l \in L(T)$ do
5:             $fs, es, n \leftarrow 0$; $r \leftarrow false$
6:             for $j \leftarrow length(\iota) - 1, 0$ do
7:                 if GOAL-P$(\iota(j))$ then
8:                     $es \leftarrow es + (er^*(l, \iota(j)) \times \iota(j)_{len})$
9:                     $fs \leftarrow fs + (fr^*(l, \iota(j)) \times \iota(j)_{len})$
10:                    $n \leftarrow n + \iota(j)_{len}$
11:                else
12:                    if $l \in$ POST$(\iota(j))$ then
13:                        $es \leftarrow es + 1$
14:                        $r \leftarrow false$
15:                    else if $l \in$ PRE$(\iota(j))$ then
16:                        $r \leftarrow true$
17:                    if $r$ then
18:                        $fs \leftarrow fs + 1$
19:                    $n \leftarrow n + 1$
20:            $Q(l, \iota) \leftarrow (fs/n, es/n, n)$
21:    return $Q$
```

Algorithm 4 Return the probability of achieving m top-level goals

```
1:  function GOALS-ACHIEVED($I$, $Q$, $m$)
2:      $p \leftarrow 0$
3:      for $X \in \{Y \subseteq I : |Y| = m\}$ do
4:          $p_X \leftarrow 1$
5:          for $\iota_i, \iota_j \in X, \iota_i \neq \iota_j$ do
6:              for $l \in L(T)$ do
7:                  $fs_i \leftarrow Q_{fs}(l, \iota_i) \times Q_n(\iota_i)$
8:                  $es_j \leftarrow Q_{es}(l, \iota_j) \times Q_n(\iota_j)$
9:                  $c^l_{i,j} \leftarrow 1 - \frac{\binom{length(\iota_j)-es_j}{fs_i}}{\binom{length(\iota_j)}{fs_i}}$
10:             $p_X \leftarrow p_X \times (1 - c^l_{i,j})$
11:         $p \leftarrow \max(p, p_X)$
12:     return $p$
```

the average length of executions of ι_i, $Q_n(\iota_i)$, and the number of establishing steps in ι_i, es_j, is given by the establishing ratio for ι_j, $Q_{er}(l, \iota_j)$ times the average length of executions of ι_j, $Q_n(\iota_j)$ (lines 7–8). We assume the worst case, that is, the fs_i steps are contiguous (constitute a single fragile region where l is established at the start of the region and is used at the end), and that the fragile region is at the start of intention ι_i. We model ι_j as a "bag" of steps, with probability $es_j/length(\iota_j)$ that the next step in ι_j is establishing. If we are equally likely to chose a step from either intention when forming the interleaving, then the probability that a step in ι_i that is fragile on l will be clobbered by an establishing step in ι_j, $c^l_{i,j}$ is the probability of placing fs_i steps in the interleaving without drawing an establishing step from ι_j (line 9). The probability that a set X of intentions is conflict free, p_X, is simply the probability that for each pair of intentions $\iota_i, \iota_j \in X$ and literal $l \in L(T)$ no conflict occurs (line 10), and we return the maximum p_X (lines 11–12).

In the *back-propagation* phase, the value is back-propagated from n_s to all nodes on the path to the root node n_0 and the visit counts of the nodes on this path are incremented. Note that we increment the visit count for n_s and its parent nodes regardless of whether any runtime simulations were actually performed from n_s. That is, if the probability of achieving m goals estimated using qualitative summary information is greater than γ, we effectively treat m as the value we would have obtained had we performed runtime simulation.

After α iterations, the step leading to the child of the root node n_0 which has the largest average simulation value (i.e., the largest number of goals achieved compared to its siblings) is returned, and the goal-plan trees and their associated current step pointers (one of which has been updated) are assigned to I for use at the next deliberation cycle.

5 EVALUATION

In this section, we evaluate the performance of S_Q in a range of scenarios of increasing difficulty. We compare the performance of S_Q with that of MCTS. The MCTS-based scheduler used in our comparison is essentially the same as the *SA* scheduler described in [35], except that it does not use coverage information to prioritise intentions which are more likely to become unexecutable in a dynamic environment, and the utility function considers only the number of goals achieved (as in S_Q) and does not consider fairness. We evaluate the performance of S_Q and MCTS on the number of goals achieved, the number of runtime simulations required, and the computation time required for each approach.

5.1 Experimental Setup

In the interests of generality, we evaluate S_Q using sets of randomly-generated, synthetic goal-plan trees representing the current intentions of an agent in a simple environment. The synthetic trees are similar to those used in the Intention Progression Competition[1] [13] and in [35], except that the trees are not always balanced, i.e., not all the leaf actions occur at the same depth. We conjecture that our estimates of fragile and establishing ratios will give more accurate results on perfectly balanced trees, since the average length of an intention will be the same as the actual length. This may result in enhanced performance of our approach on perfectly balanced trees, and, in order to give unbiased comparison with the performance of MCTS on arbitrary trees, we avoided the use of balanced trees.

The unbalanced trees were generated by introducing a new parameter, x, that specifies the probability of a plan being a leaf plan. That is, rather than only generating leaf plans when the maximum depth of the goal-plan tree is reached, a plan has probability x of being a leaf plan, even when the current depth is smaller than the maximum depth of the tree. An issue with unbalanced trees is that the agent will favour short paths. To balance the difficulty of executions of different length, we increased the number of preconditions on short paths, i.e., short paths are more likely to cause conflicts when interleaved with other intentions. This seems reasonable — in many cases the simplicity of a plan is inversely proportional to, e.g., its resource cost.

In the experiments reported below, we vary the maximum depth, d, of the goal-plan trees from 4 to 6, each goal has two relevant plans, and each plan contains three actions and two subgoals. The probability of a plan being a leaf plan was set to 0.2. The agent's environment is built from 60 propositions (corresponding to 120 literals). For each goal-plan tree, we select 30 propositions at random and choose from the corresponding literals the pre- and postconditions of the actions in the tree. In each experiment, we generate 50 sets of 10 goal-plan trees with the parameters specified above.[2] The fr^* and er^* ratios were calculated offline, by running 10000 random simulations from each plan in each goal-plan tree.

The α parameter specifying the number of nodes expanded at each deliberation cycle was set to 100 as in [35]. The β parameter specifying the number of runtime simulations performed by MCTS (and that may be performed by S_Q) was varied to create different computational settings. S_Q was configured with $\gamma = 0.5$ and $\delta = 0.1$, i.e., we only perform runtime simulations when the probability of achieving m goals is in the range (0.1, 0.5). In all other cases, we trust the quantitative summary information.

5.2 Experimental Results

To investigate whether the use of quantitative summary information by S_Q reduced the number of runtime simulations performed

[1] https://www.intentionprogression.org/
[2] The source code for S_Q and goal-plan trees used in the experiments are available at https://www.github.com/yvy714/SQ-MCTS.git

d	(α, β)		(100, 100)	(100,10)	(100,1)
4	MCTS	#goals	9.9	9.7	9
		#sims	2.27×10^6	2.24×10^5	2.36×10^4
		time	1406.1	289.6	45.2
	S_Q	#goals	9.9	9.8	9.2
		#sims	1.04×10^6	1.10×10^5	1.14×10^4
		#saved	1.23×10^6	1.13×10^5	1.22×10^4
		time	899.7	229.6	185.1
5	MCTS	#goals	8.9	8.7	7.7
		#sims	2.82×10^6	2.73×10^5	2.85×10^4
		time	1603.4	352.0	55.6
	S_Q	#goals	9.6	9.1	8.9
		#sims	1.22×10^6	1.13×10^5	1.23×10^4
		#saved	1.60×10^6	1.60×10^5	1.61×10^4
		time	1002.0	276.0	213.1
6	MCTS	#goals	7.5	6.6	5.6
		#sims	4.04×10^6	3.85×10^5	3.07×10^4
		time	1737.1	415.2	66.4
	S_Q	#goals	8.0	7.6	6.4
		#sims	1.47×10^6	1.70×10^5	1.27×10^4
		#saved	2.57×10^6	2.14×10^5	1.80×10^4
		time	1121.1	317.8	230.9

Table 1: Goals achieved and runtime simulations with varying computational budget and problem difficulty.

and/or computation time, we considered three computational budgets: $\beta = 100$, $\beta = 10$, and $\beta = 1$, and three levels of problem difficulty: $d = 4$, $d = 5$, and $d = 6$. In Table 1, we report the average (over 50 runs) number of goals achieved (#goals), average number of runtime simulations required (#sims), and the average computation time for each approach in milliseconds (time). This is the time required to return the first action to be executed, i.e., to compute a complete interleaving of actions in all intentions starting from the root of each goal-plan tree, and is essentially the worst case for both approaches. As noted in [35], in a static environment, this interleaving only needs to be recomputed when the agent adopts a new top-level goal, so in the best case (no new top-level goals during the execution of the current intentions) the computation time is effectively amortised over the execution of all 10 intentions. For S_Q, we also report the average saving in runtime simulations (#saved).[3]

As can be seen, for goal-plan trees with a maximum depth of 4, the number of goals achieved by both approaches declines as the computational budget (number of runtime simulations) decreases. However, S_Q achieves at least as many goals as MCTS in all cases, and requires approximately 50% of the runtime simulations required by MCTS. (The $(\alpha = 100, \beta = 10)$ case represents the same computational budget used in [35], and the number of goals achieved in this case indicates that the unbalanced trees with maximum depth 4 are of similar difficulty to the balanced trees of depth 5 used in [35] to evaluate *SA*.) In the $\beta = 100$ and $\beta = 10$ cases, the reduction in the number of runtime simulations results in a reduction in computation time of approximately 36% and 21% respectively. In the $\beta = 1$ case, the overhead of computing qualitative summary information is significantly greater than the computation time saved by the reduction in runtime simulations. However, with a small number runtime simulations, the performance of MCTS can be unstable, i.e., not only is the average solution quality lower, the variance in the quality of solution returned is greater.

For goal-plan trees with maximum depth 5, the performance of both approaches declines. (At this depth, the goal-plan trees require about six times as many steps to achieve a top-level goal as the GPTs in [35].) However the decline is more marked in the case of MCTS, particularly for smaller values of β. Moreover, S_Q requires 43% and 41% of the runtime simulations required by MCTS in the $\beta = 100$ and $\beta = 10$ cases, with a corresponding reduction in computation time of 38% and 22%. As in the $d = 4$ case, when $\beta = 1$, the overhead of computing qualitative summary information is significantly greater than the computation time saved by the reduction in runtime simulations. However, S_Q can achieve approximately the same number of goals with one simulation as MCTS with 10 simulations, and in 61% of the computation time, indicating that S_Q is effectively exploiting qualitative summary information. At $d = 6$, a similar pattern can be observed. The performance of both approaches declines further, with a more marked reduction in the case of MCTS. S_Q requires only 36% and 44% of the runtime simulations required by MCTS, with a corresponding reduction in computation time of 36% and 24% in the $\beta = 100$ and $\beta = 10$ cases. When $\beta = 1$, S_Q can achieve approximately the same number of goals as MCTS with $\beta = 10$, and in 56% of the computation time.

Our experiments did not consider the case of a dynamic environment. However, we stress that S_Q can handle exogenous events. The fr^* and er^* values for GPTs are unaffected by changes to the environment, and the QSI and GOALS-ACHIEVED values are recomputed at each cycle based on the actual progression of each intention (which in turn is based on the agent's current beliefs).

6 RELATED WORK

In addition to the work of Thangarajah et al. [21–23] and Yao et al. [35–37] discussed above, a number of other approaches to scheduling intentions to avoid conflicts have been proposed in the literature.

Waters et al. [29, 30] present a *coverage-based* approach to intention selection proposed by [23], in which the intention with the lowest coverage, i.e., the highest probability of becoming non-executable due to changes in the environment, is selected for execution. As in [21–23], intention selection is limited to the plan level, and their experimental evaluation assumes that there are no conflicts between intentions. Shaw and Bordini have proposed approaches to intention selection based on Petri nets [18] and constraint logic programming [19]. Again, as in [21–23], the plans and sub-goals in a goal-plan tree are regarded as basic steps, and interleaving is at the level of sub-plans and subgoals. They do not consider interactions between actions in plans.

The TÆMS (Task Analysis, Environment Modelling, and Simulation) framework [9] together with Design-To-Criteria (DTC) scheduling [27] have been used in agent architectures such the

[3]The average number of simulations and simulations saved are rounded to two significant figures.

Soft Real-Time Agent Architecture [25] and AgentSpeak(XL) [1] to schedule intentions. TÆMS provides a high-level framework for specifying the expected quality, cost and duration of of methods (actions) and relationships between tasks (plans). DTC decides which tasks to perform, how to perform them, and the order in which they should be performed, so as to satisfy hard constraints (e.g., deadlines) and maximise the agent's objective function. DTC can produce schedules which allow interleaved or parallel execution of tasks and can be used in an anytime fashion. In the work closest to that presented here [1], DTC was used to schedule execution of AgentSpeak intentions at the level of individual plans. The TÆMS relations between plans required to generate a schedule (*enables*, *facilitates* and *hinders*) were specified as part of the agent program. In contrast S_Q interleaves intentions at the level of actions, and information about possible conflicts between intentions is extracted automatically from goal-plan trees generated from the agent program.

In [15] Sardina et al. show how an HTN planner can be integrated into a BDI agent architecture. However their focus is on finding a hierarchical decomposition of a plan that is less likely to fail by avoiding incorrect decisions at choice points, and they do not take into account interactions with other concurrent intentions. In [31], Wilkins et al. presented the Cypress architecture which combines the the Procedural Reasoning System reactive executor PRS-CL, and the SIPE-2 look-ahead planner. A Cypress agent uses PRS-CL to pursue its intentions using a library of procedures (plans). If a failure occurs during the execution of the plan due to an unanticipated change in the agent's environment, the executor calls SIPE-2 to produce a new plan to achieve the goal, and continues executing those portions of plans which are not affected. However, their approach focusses on the generation of new plans to recover from plan failures, rather than interleaving intentions so as to avoid conflicts.

Among other approaches to deliberation about conflicts between plans is the abstract programming language developed in [8] which is inspired by the BOID architecture. In [28], an approach to runtime conflict resolution between goals based on event calculus is proposed and experimentally evaluated.

There has also been work on avoiding conflicts in a multi-agent setting. For example, Clement and Durfee [4–6] propose an approach to coordinating concurrent hierarchical planning agents using summary information and HTN planning. However in this work, summary information is used to identify when conflicts may arise between two or more agents rather than to avoid conflicts between the intentions of a single agent. Moreover, it is assumed that the agents plan offline in a static environment. In [11], Ephrahi et al. present an approach to planning and interleaving the execution of tasks by multiple agents. The task of each agent is assigned dynamically, and the execution of all tasks achieves a global goal. They show how conflicts between intentions can be avoided by appropriate scheduling of the actions of the agents. In [7] Dann et al. extend the MCTS-based approach of Yao et al. [35] to a multi-agent setting, in which agents consider not only interactions between their own intentions but the intentions of other agents.

Visser et al. [26] have used summary information to reason about preferences in BDI agents. They specify preferences over resources and properties (e.g., payment-type, flight-class, hotel-rating etc.), and use resource and property summary information to deliberate over the selection of plans and the ordering of subgoals in a plan when the ordering is not fixed by design, in order to maximise the satisfaction of the agent's preferences. Their approach however, does not consider preference satisfaction across multiple intentions.

Many approaches to enhancing the performance of MCTS have been proposed in the literature, both to reduce the search space (for example pruning, analogously to α-β-pruning [10, 17]), and to improve the performance of simulation, such as domain-specific modifications to the default policy, backpropagation policy, parallelisation, etc. [3]. Our approach does not change the MCTS search space. Rather, it falls under enhancements to the simulation phase, since during simulation we use information obtained off-line in order avoid performing unnecessary computation. We are not aware of similar approaches to simulation performance enhancement in the MCTS literature. However, our approach can be used in combination with techniques such as [3, 10, 17] .

7 CONCLUSION

In this paper, we introduced the notion of quantitative summary information which can be used to estimate the likelihood of conflicts between an agent's intentions. We showed how offline simulation can be used to precompute quantitative summary information prior to execution of the agent's program, and how the precomputed summary information can be used at runtime to guide the expansion of the MCTS search tree and avoid unnecessary runtime simulation. We compared the performance of our approach with standard MCTS in a range of scenarios of increasing difficulty. The results suggest our approach can reduce the number of runtime simulations performed by up to 64% and the time required to schedule an agent's intentions by up to 38%.

In its current form, S_Q considers only the non-executability of intentions due to conflicts. In future work we plan to investigate whether notions similar to coverage [23] can be incorporated into quantitative summary information to estimate the probability of unestablished fragile literals (i.e., unestablished preconditions) resulting in the non-executability intentions.

ACKNOWLEDGMENTS

This work was supported by Zhejiang Provincial Natural Science Foundation of China (LQ19F030009) and National Natural Science Foundation of China (61906169).

REFERENCES

[1] Rafael Bordini, Ana L. C. Bazzan, Rafael de O. Jannone, Daniel M. Basso, Rosa M. Vicari, and Victor R. Lesser. 2002. AgentSpeak(XL): Efficient Intention Selection in BDI Agents via Decision-Theoretic Task Scheduling. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*. 1294–1302.

[2] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. 2007. *Programming multi-agent systems in AgentSpeak using Jason.* Wiley.

[3] Cameron Browne, Edward J. Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (March 2012), 1–43.

[4] Bradley J. Clement and Edmund H. Durfee. 1999. Theory for Coordinating Concurrent Hierarchical Planning Agents Using Summary Information. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence* (Orlando, Florida,

USA), Jim Hendler and Devika Subramanian (Eds.). AAAI Press / The MIT Press, 495–502.

[5] Bradley J. Clement and Edmund H. Durfee. 2000. Performance of Coordinating Concurrent Hierarchical Planning Agents Using Summary Information. In *4th International Conference on Multi-Agent Systems* (Boston, MA, USA). IEEE Computer Society, 373–374.

[6] Bradley J. Clement, Edmund H. Durfee, and Anthony C. Barrett. 2007. Abstract Reasoning for Planning and Coordination. *Journal of Artificial Intelligence Research* 28 (2007), 453–515.

[7] Michael Dann, John Thangarajah, Yuan Yao, and Brian Logan. 2020. Intention-Aware Multiagent Scheduling. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020)* (Auckland), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, and G. Sukthankar (Eds.). IFAAMAS, 285–293.

[8] Mehdi Dastani and Leendert W. N. van der Torre. 2004. Programming BOID-Plan Agents: Deliberating about Conflicts among Defeasible Mental Attitudes and Plans. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)* (New York, NY). IEEE Computer Society, 706–713.

[9] Keith S. Decker and Victor R. Lesser. 1993. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance and Management* 2 (1993), 215–234.

[10] Joris Duguépéroux, Ahmad Mazyad, Fabien Teytaud, and Julien Dehos. 2016. Pruning Playouts in Monte-Carlo Tree Search for the Game of Havannah. In *Proceedings of the 9th International Conference on Computers and Games (CG 2016)* (Leiden) *(Lecture Notes in Computer Science, Vol. 10068)*, Aske Plaat, Walter A. Kosters, and H. Jaap van den Herik (Eds.). Springer, 47–57.

[11] Eithan Ephrati and Jeffrey S. Rosenschein. 1993. A Framework for the Interleaving of Execution and Planning for Dynamic Tasks by Multiple Agents. In *From Reaction to Cognition, 5th European Workshop on Modelling Autonomous Agents, MAAMAW '93* (Neuchatel, Switzerland), Cristiano Castelfranchi and Jean-Pierre Müller (Eds.). Springer, 139–153.

[12] Jana Koehler and Kilian Schuster. 2000. Elevator Control as a Planning Problem. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000* (Breckenridge, CO, USA), Steve Chien, Subbarao Kambhampati, and Craig A. Knoblock (Eds.). AAAI, 331–338.

[13] Brian Logan, John Thangarajah, and Neil Yorke-Smith. 2017. Progressing Intention Progression: A Call for a Goal-Plan Tree Contest. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)*, (Sao Paulo, Brazil), S. Das, E. Durfee, K. Larson, and M. Winikoff (Eds.). IFAAMAS, 768–772.

[14] Anand S. Rao and Michael P. Georgeff. 1992. An Abstract Architecture for Rational Agents. In *3rd International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann, 439–449.

[15] Sebastian Sardiña, Lavindra de Silva, and Lin Padgham. 2006. 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)* (Hakodate, Japan), Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone (Eds.). ACM, 1001–1008.

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

[17] Nick Sephton, Peter I. Cowling, Edward Jack Powley, and Nicholas H. Slaven. 2014. Heuristic move pruning in Monte Carlo Tree Search for the strategic card game Lords of War. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2014)*. 1–7.

[18] Patricia H. Shaw and Rafael H. Bordini. 2007. Towards Alternative Approaches to Reasoning About Goals. In *Declarative Agent Languages and Technologies V, 5th International Workshop* (Honolulu, HI, USA), Matteo Baldoni, Tran Cao Son, M. Birna van Riemsdijk, and Michael Winikoff (Eds.), Vol. 4897. Springer, 104–121.

[19] Patricia H. Shaw and Rafael H. Bordini. 2010. An Alternative Approach for Reasoning about the Goal-Plan Tree Problem. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)* (Lisbon, Portugal), Helder Coelho, Rudi Studer, and Michael Wooldridge (Eds.), Vol. 215. IOS Press, 1035–1036.

[20] Patricia H. Shaw, Berndt Farwer, and Rafael H. Bordini. 2008. Theoretical and experimental results on the goal-plan tree problem. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Lin Padgham, David C. Parkes, Jörg P. Müller, and Simon Parsons (Eds.). IFAAMAS, 1379–1382.

[21] John Thangarajah and Lin Padgham. 2011. Computationally Effective Reasoning About Goal Interactions. *Journal of Automated Reasoning* 47, 1 (2011), 17–56.

[22] John Thangarajah, Lin Padgham, and Michael Winikoff. 2003. Detecting & Avoiding Interference Between Goals in Intelligent Agents. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)* (Acapulco, Mexico), Georg Gottlob and Toby Walsh (Eds.). Morgan Kaufmann, 721–726.

[23] John Thangarajah, Sebastian Sardina, and Lin Padgham. 2012. Measuring plan coverage and overlap for agent reasoning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)* (Valencia, Spain), Wiebe van der Hoek, Lin Padgham, Vincent Conitzer, and Michael Winikoff (Eds.). IFAAMAS, 1049–1056.

[24] Leslie G. Valiant. 1979. The Complexity of Computing the Permanent. *Theoretical Computer Science* 8, 2 (1979), 189–2001.

[25] Régis Vincent, Bryan Horling, Victor Lesser, and Thomas Wagner. 2001. Implementing Soft Real-Time Agent Control. In *Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS'01)* (Montreal, Quebec, Canada). ACM Press, New York, NY, USA, 355–362.

[26] Simeon Visser, John Thangarajah, James Harland, and Frank Dignum. 2016. Preference-based reasoning in BDI agent systems. *Autonomous Agents and Multi-Agent Systems* 30, 2 (2016), 291–330.

[27] Thomas Wagner, Alan Garvey, and Victor Lesser. 1998. Criteria-Directed Heuristic Task Scheduling. *International Journal of Approximate Reasoning* 19 (1998), 91–118.

[28] Xiaogang Wang, Jian Cao, and Jie Wang. 2012. A Runtime Goal Conflict Resolution Model for Agent Systems. In *2012 IEEE/WIC/ACM International Conferences on Intelligent Agent Technology, IAT 2012*. IEEE Computer Society, 340–347.

[29] Max Waters, Lin Padgham, and Sebastian Sardina. 2014. Evaluating Coverage Based Intention Selection. In *Proceedings of the 13th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2014)* (Paris, France), Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (Eds.). IFAAMAS, 957–964.

[30] Max Waters, Lin Padgham, and Sebastian Sardiña. 2015. Improving domain-independent intention selection in BDI systems. *Autonomous Agents and Multi-Agent Systems* 29, 4 (2015), 683–717. https://doi.org/10.1007/s10458-015-9293-5

[31] David E. Wilkins, Karen L. Myers, John D. Lowrance, and Leonard P. Wesley. 1995. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical Artificial Intelligence* 7, 1 (1995), 121–152.

[32] Michael Winikoff. 2005. JACK Intelligent Agents: An Industrial Strength Platform. In *Multi-Agent Programming: Languages, Platforms and Applications*, Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni (Eds.). Multiagent Systems, Artificial Societies, and Simulated Organizations, Vol. 15. Springer, 175–193.

[33] Michael Winikoff and Stephen Cranefield. 2014. On the Testability of BDI Agent Systems. *Journal of Artificial Intelligence Research* 51 (2014), 71–131.

[34] Yuan Yao, Lavindra de Silva, and Brian Logan. 2016. Reasoning about the Executability of Goal-Plan Trees. In *Proceedings of the 4th International Workshop on Engineering Multi-Agent Systems (EMAS 2016) (LNCS, Vol. 10093)*, Matteo Baldoni, Jorg P. Muller, Ingrid Nunes, and Rym Zalila-Wenkstern (Eds.). Springer, 176–191.

[35] Yuan Yao and Brian Logan. 2016. Action-Level Intention Selection for BDI Agents. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)* (Singapore), J. Thangarajah, K. Tuyls, C. Jonker, and S. Marsella (Eds.). IFAAMAS, 1227–1235.

[36] Yuan Yao, Brian Logan, and John Thangarajah. 2014. SP-MCTS-based Intention Scheduling for BDI Agents. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI-2014)* (Prague, Czech Republic), Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan (Eds.). IOS Press, 1133–1134.

[37] Yuan Yao, Brian Logan, and John Thangarajah. 2016. Robust Execution of BDI Agent Programs by Exploiting Synergies Between Intentions. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)* (Phoenix, USA), Dale Schuurmans and Michael P. Wellman (Eds.). AAAI Press, 2558–2564.