Action-Level Intention Selection for BDI Agents

Yuan Yao
School of Computer Science
University of Nottingham
Nottingham, UK
yvy@cs.nott.ac.uk

Brian Logan
School of Computer Science
University of Nottingham
Nottingham, UK
bsl@cs.nott.ac.uk

ABSTRACT

Belief-Desire-Intention agents typically pursue multiple goals in parallel. However the interleaving of steps in different intentions may result in conflicts, e.g., where the execution of a step in one plan makes the execution of a step in another concurrently executing plan impossible. Previous approaches to avoiding conflicts between concurrently executing intentions treat plans as atomic units, and attempt to interleave plans in different intentions so as to minimise conflicts. However some conflicts cannot be resolved by appropriate ordering of plans and can only be resolved by appropriate interleaving of steps within plans. In this paper, we present *SA*, an approach to intention selection based on Single-Player Monte Carlo Tree Search that selects which intention to progress at the current cycle at the level of individual plan steps. We evaluate the performance of our approach in a range of scenarios of increasing difficulty in both static and dynamic environments. The results suggest *SA* out-performs existing approaches to intention selection both in terms of goals achieved and the variance in goal achievement time.

1. INTRODUCTION

In Belief-Desire-Intention-based (BDI) agent programming languages, e.g., [4, 9], 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. *Plans* are the means by which the agent can modify the environment in order to achieve its goals. Plans are composed of *steps* which are either *primitive actions* that directly change the agent's environment or *subgoals* which are in turn achieved by other plans. For each top-level goal, the agent selects a plan which forms the root of an *intention*, and commences executing the steps in the plan. If the next step in an intention is a subgoal, a (sub)plan is selected to achieve the subgoal and added to the intention, and the steps in the (sub)plan are then executed and so on. This process of repeatedly choosing and executing plans is referred to as the agent's *deliberation cycle*.

In many BDI agent architectures, the plans comprising the agent's intentions are executed in parallel, e.g., by executing one step of an intention at each cycle in round robin fashion [4, 30]. Interactions between interleaved steps in different intentions may result in conflicts, i.e., the execution of a step makes it impossible to execute a step in another concurrently executing intention. For example, consider a Mars Rover which has a goal to perform a rock experiment, r, and a goal to upload some image data, u. The plan to perform a rock experiment consists of two actions: collecting a sample, a_1, and analysing the sample, a_2. $a1$ has precondition p (being at the appropriate location) and a_2 has the postcondition $\neg q$ (overwriting the current contents of the rover's data buffer). The plan to upload image data also consists of two steps: compressing the image, b_1, and transmitting the compressed data, b_2. b_1 has precondition q (the image currently in the data buffer), and b_2 has postcondition $\neg p$ (the rover must be reoriented for transmission). Even if p and q have been established by previous steps in the intentions for r and u, it is clear that r and u cannot both be achieved without an appropriate interleaving of plan steps.

A number of approaches to reasoning about possible interactions between intentions have been proposed in the literature. Thangarajah et al. [22, 21] describe an approach based on *summary information* that avoids conflicts by reasoning about necessary and possible pre- and post-conditions of different ways of achieving a goal. They give algorithms for computing summary information at compile time, and for dynamically updating it at run-time. They also present mechanisms to determine whether a newly adopted (sub)goal will definitely be safe to execute without conflicts, or will definitely result in conflicts, or may result in conflicts. If the goal cannot be executed safely, execution of the intention is deferred. Yao et al. [31] present a stochastic approach based on Single-Player Monte-Carlo Tree Search (SP-MCTS) in which pseudorandom simulations of different interleavings of the plans in each intention are used to determine which intention to progress. However these approaches are limited to interleaving intentions at the plan level, i.e., execution of another intention is only possible when a subgoal is posted and steps in different plans are not interleaved, and they have only been evaluated in a static environment. Waters et al. [27, 28] 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. Unlike Thangarajah et al. and Yao et al. [23, 31], they evaluate their approach in a dynamic environment; however intention selection is again limited to the plan level, and their experimental evaluation assumes that there are no conflicts between intentions.

In this paper, we introduce *SA*, a new approach to avoiding conflicts between intentions. *SA* extends the stochastic scheduling approach of Yao et al. [31] in two ways. First, we allow the interleaving of primitive actions in different intentions. Second, we change the function used to select the 'best' intention to progress at this cycle to take both the dynamism of the environment and the fairness of the interleaving into account. We evaluate the performance of *SA* and compare it to that of round robin (RR), first in first out (FIFO), summary information-based (SI) and coverage-based (C) intention

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

selection in static and dynamic environments, both in synthetic domains and a more realistic scenario based on the Miconic-10 elevator domain [14]. Our results suggest *SA* out-performs RR, FIFO, SI, and C both in terms of goals achieved and/or the variance in goal achievement time, and is capable of successfully interleaving intentions so as to achieve their top-level goals even when the number of potential conflicts between intentions is high. Moreover, in the scenarios considered, the computational overhead of *SA* is modest.

2. GOAL-PLAN TREES

We use the notion of *goal-plan trees* [22, 21] to represent the relations between goals, plans and actions, and to reason about the interactions between intentions. The root of a goal-plan tree is a top-level goal (goal-node), and its children are the plans that can be used to achieve the goal (plan-nodes). Plans may in turn contain subgoals (goal nodes), giving rise to a tree structure representing all possible ways an agent can achieve the top-level goal. In [22, 21] goal-plan trees contain only goals and plans. To support action-level intention selection, we extend the definition of goal-plan trees to allow primitive actions in plans in addition to subgoals as in [32].

To allow reasoning about interactions between intentions, each 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-, in- 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. *In-conditions* are conditions that must hold during the pursuit of a goal or plan; if an in-condition becomes false during the execution of a goal or plan, the goal or plan is dropped with failure. *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).

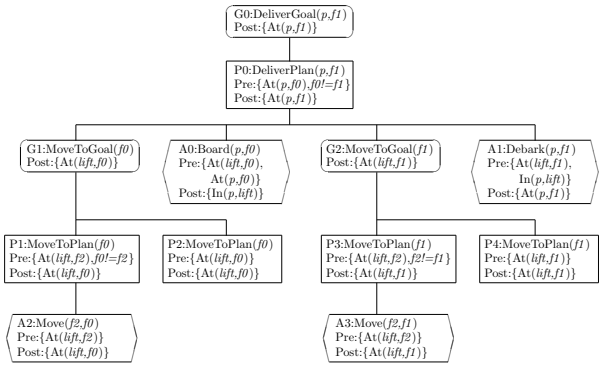

Figure 1: A goal-plan tree for the elevator domain

Figure 1 shows a simple goal-plan tree for the elevator domain [14]. The top level goal, G0, is to deliver passengers *p* to floor *f1*. There is a single plan, P0, to achieve G0. P0 consists of two subgoals, G1 and G2 and two actions, A0 and A1. Each subgoal may be achieved by two subplans (P1 and P2 for G1 and P3 and P4 for G2). The pre- and postconditions of plans and actions are shown in the figure; for example, the action A0:Board(*p, f0*), has precondition that both the *lift* and passengers *p* are at floor *f0*, and postcondition that the passengers are in the lift.

In addition to defining the relationships between goals, plans and actions, goal-plan trees can be used to define several notions that characterise the robustness of an agent program in a given environment (and hence the likelihood that a particular interleaving of two goal plan trees will result in conflicts). The *coverage* of a goal-plan tree is the percentage of world states for which there is some applicable plan for any subgoal within an intention. The *overlap* of a goal-plan tree is the percentage of world states in which two or more plans for a goal are applicable [23]. A preparatory-effect or *p-effect* occurs when a plan or action establishes the precondition of a later step (subplan or action) in the same goal-plan tree [22]. (Note that 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.) For example, in Figure 1, a precondition of action A0, At(*lift, f0*), is established by executing plans P1 or P2. At(*lift, f0*) is therefore a p-effect which needs to be protected until action A0 is executed. We define the *size of a p-effect* as the number of possible interleaving points between the plan or action that establishes a precondition and the later step which requires the p-effect as its precondition. For example, the size of the p-effect At(*lift, f0*) is 1, as there is only one point at which a step from another intention can be interleaved. Like coverage and overlap, p-effect size is a measure of an intention's robustness: the greater size of the p-effects, the more likely interleaving steps from other intentions will destroy (clobber) the established precondition.

In a dynamic environment, *SA* takes the coverage of each goal-plan tree into account when choosing which intention to progress. We also use coverage, overlap and p-effect size to characterise the difficulty of the evaluation scenarios in Section 6.

3. SELECTING INTENTIONS

We represent the agent's intentions as a set of goal-plan trees $T = \{t_1, \ldots, t_n\}$ corresponding to the agent's top-level goals. The progression of an intention to achieve a top-level goal amounts to choosing a path through the corresponding goal-plan tree. Each path corresponds to a different way of achieving the top level goal forming the root of the tree. The path specifies a sequence of plans, actions, subgoals and sub-plans that, if executed successfully, will achieve the top-level goal. The execution of an agent program thus corresponds to an interleaving of paths through each of the goal-plan trees corresponding to the agent's current intentions.

An inappropriate interleaving of plans or primitive actions in different goal-plan trees can give rise to conflicts. We say the progression of a set of intentions is *conflict free*, if the choice of path through each goal-plan tree, and the interleaving of the steps in these paths, ensure that the preconditions of each plan and action on a path are true when the plan or action are executed, and that any in-conditions required by a goal or plan are true during the achievement of the goal or execution of the plan.

To minimise conflicts (and hence maximise the number of goals achieved),[1] an intention selection mechanism must be able to appropriately interleave the execution of steps in an agent's intentions. In many cases, the interleaving of intentions is also desirable in its own right, e.g., to ensure more predictable response times for users or other agents. We therefore prefer interleavings that maximise *fairness*. We say an interleaving is more fair if it minimises the variance in the elapsed time of each successful intention (subject to achieving as many goals as possible). The elapsed time, r, of an intention is defined as $r = d_f - d_a$, where d_a is the cycle at which the top-level goal of the intention is adopted by the agent, and d_f is the cycle at which the goal is achieved. The *variance in elapsed time*, v, of a set of n intentions with elapsed times $r_1, \ldots, r_n$ is given by $v = \sum_{i=1}^{n} (r_i - \overline{r})^2 / n$, where $\overline{r}$ is the average elapsed time. Fairness is then defined as $1/v$.

[1]We implicitly assume that goals are of equal importance; it is straightforward to modify our approach to handle cases where some goals are more important than others.

The *intention selection problem* is the problem of choosing an intention to progress at the current deliberation cycle, such that the progression of the set of intentions is conflict free. To make the problem precise, we introduce the notion of the current step, s_i, and next step, $succ(s_i)$, of a goal-plan tree t_i representing an intention. s_i is a pointer to the current step in t_i, which may be a primitive action or a subgoal. The path from the root of the goal-plan tree to the current step represents the choices made (and steps executed) so far in achieving the top-level goal. When an agent adopts an intention for a new top-level goal g_i, the current step of the goal-plan tree rooted at g_i is initially set to g_i itself. The next step $succ(s_i)$ specifies how t_i is to be progressed following s_i. If the current step is a primitive action, the next step is the primitive action or subgoal following that action in the same plan. If the current step is a subgoal, advancing the current step involves choosing a plan for the subgoal, and setting the next step to be the first action or subgoal of the selected plan. The intention selection problem can now be stated as: given a set of goal plan trees and their current step pointers $I = \{(t_1, s_1), \ldots, (t_n, s_n)\}$ corresponding to the agent's current intentions, and a set of condition variables E representing the current state of the agent's environment, return a next step $succ(s_i)$ of an intention $(t_i, s_i) \in I$ to be executed at the current deliberation cycle. We further require that $succ(s_i)$ maximises the number of goals achieved and fairness of the interleaving, subject to achieving as many goals as possible. That is, there is no $(t_j, s_j) \in I, j \neq i$ such that progressing t_j would achieve more goals than progressing t_i, or if progressing t_j and t_i achieve the same number of goals, progressing t_j results in an interleaving that is at least as fair.

4. SA ALGORITHM

Our approach to intention scheduling is based on Single-Player Monte-Carlo Tree Search (SP-MCTS) [18]. SP-MCTS is a best-first search in which pseudorandom simulations are used to guide expansion of the search tree. It was originally developed to solve single-player puzzles (games against the environment) [18], however it has also been used successfully to solve reentrant scheduling problems [15].

Algorithm 1 Return the action be executed at this cycle

```
1:  I ← {(t_1, s_1), ..., (t_n, s_n)}
2:  procedure SA(E, e, δ, α, β)
3:      n_0 ← node_0(I, E)
4:      for i ← 1, α do
5:          n_e ← MAX-UCT-LEAF-NODE(n_0)
6:          children(n_e) ← EXPAND(n_e)
7:          n_s ← RANDOM-CHILD(children(n_e))
8:          S ← ∅
9:          for j ← 1, β do
10:             S ← S ∪ {SIMULATE(n_s)}
11:         v_g(n_s), v_f(n_s) ← max_{<_{g,f}}(S)
12:         BACKUP(n_s)
13:     if e > δ then
14:         n_b ← max_{<_{g,c,f}} children(n_0)
15:     else
16:         n_b ← max_{<_{g,f}} children(n_0)
17:     I ← gpt(n_b)
18:     return (n_0, n_b)
```

The *SA* algorithm updates the set of goal plan trees and their current step pointers $I = \{(t_1, s_1), \ldots, (t_n, s_n)\}$ corresponding to the agent's current intentions. It takes five parameters as input: the current state of the agent's environment, E, the average number of environment changes per deliberation cycle, e, the dynamism threshold, δ, the number of iterations to be performed, α, and the number of simulations to be performed at each iteration, β; and returns a next step of an intention in I for execution at this cycle (see Algorithm 1). The parameters δ, α and β allow the agent developer to configure *SA* for a particular application: δ specifies the mean number of environment changes per deliberation cycle at which the agent should favour more robust (higher coverage) plans; α and β determine the effort expended on intention selection.

Starting from a root node n_0 representing the current progression of the agent's intentions I and state of the environment E, the algorithm iteratively builds a search tree. Each node in the search tree represents an interleaving of steps from the goal-plan trees in I that is consistent with the order of steps in each tree. The node records the state of the agent's environment resulting from the execution of this interleaving, the current step in each goal-plan tree, the start time of all top-level goals and the completion time for the goals achieved by the interleaving. In addition, the node also contains a record of the number of times it has been visited, the value of the node, and the best simulation performed from the node (described below). Edges represent the selection of a plan for a subgoal or the execution of primitive action in a plan.

Each iteration of the main loop (lines 4–12) consists of 4 phases: selection, expansion, simulation and back-propagation. In the *selection* phase, a leaf node, n_e, is selected for expansion (line 5). A node may be expanded if it represents a non-terminal state (a state in which it is possible to execute the next step of a goal-plan tree). n_e is selected using a modified version of Upper Confidence bounds applied to Trees (UCT) [18], which models the choice of node as a k-armed bandit problem. Starting from the root node n_0, we recursively follow child nodes with highest UCT value until a leaf node n_e is reached.

In the *expansion* phase, nodes representing the environment state (and the current step of each goal-plan tree) resulting from executing each next step of a goal-plan tree that is executable the state represented by n_e are added as children of n_e (line 6). Each child node corresponds to a different choice of which intention to execute at this cycle. One of the newly created child nodes, n_s, is then selected at random for simulation (line 7).

In the *simulation* phase, the value of n_s is estimated by performing β pseudorandom simulations (lines 8–11). The *value* of a node consists of two components: the maximum number of goals that can be achieved from this node, v_g, and the fairness of the interleaving, v_f. Each simulation starts in the environment represented by n_s. A next step of a goal-plan tree that is executable in the environment state represented by n_s is randomly selected and executed, and the environment and the current step of the selected goal-plan tree updated. We keep choosing executable next steps and updating the environment 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 values of v_g and v_f in the terminal state. The simulation with the highest value is returned as the value of n_s ($<_{g,f}$ is a lexicographic order on v_g and v_f, i.e., v_f is used to break ties when child nodes have the same v_g value).

Finally, the simulation value is *back-propagated* from n_s to all nodes on the path to the root node n_0 (line 12).

As the search tree is expanded by iteration of the main loop, the random part of the interleaving represented by leaf nodes becomes shorter, and we obtain better estimates of v_g and v_f. After α iterations, the step that leads to the best child n_b of the root node n_0 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 (lines 13–18). (The function gpt returns the goal-plan trees and their associated current step pointers represented by the node n_b.) Which step is considered best depends on the degree of dynamism of the environment. If the mean number of environment changes observed by the agent is less than the dynamism threshold set by the agent developer, δ, *SA* selects steps using $<_{g,f}$, i.e., first on the number of top-level goals achieved, and, where two or more steps result in the achievement of the same number of goals, it prefers steps that also maximise fairness. If the dynamism exceeds the threshold, *SA* selects steps using the order $<_{g,c,f}$, i.e., first on the number of goals achieved, secondly on coverage (intentions with lower coverage are preferred) and thirdly which maximise fairness.

SA therefore adopts a similar approach to [23, 27, 28] in dynamic environments, in prioritising intentions which are more likely to become unexecutable due to changes in the environment. However, unlike previous coverage-based approaches, it selects the intention to be progressed at the action, rather than plan level, interleaving actions in a manner that is most likely to achieve the largest number of goals, and adaptively prioritises fairness in low dynamism environments. Our use of SP-MCTS to sample the space of possible interleavings is similar to [31], but that work interleaves intentions at the plan level and does not take the coverage of intentions or fairness into account.

5. SA AND THE DELIBERATION CYCLE

SA implies some modifications to the standard BDI deliberation cycle. In this section we briefly sketch these changes and some obvious optimisations of our approach.

The standard BDI deliberation cycle consists of three phases [5, 16]:

1. select one or more events to process at this cycle;
2. for each selected goal event, select a relevant applicable plan to achieve the goal;
3. finally, select one or more intentions, and execute one or more steps of each selected intention.

Particular BDI languages may subdivide one or more of these phases. However some instantiation of this basic deliberation cycle is found in most BDI agent languages and platforms.

SA effectively merges elements of phase 1 and phases 2 and 3 of the standard deliberation cycle into a single process which is responsible for determining both which plan to adopt for a particular (sub)goal, and which step of which intention to execute at this cycle. Only the choice of which top-level goals to adopt falls outside the scope of *SA*: once a top-level goal is adopted, *SA* determines which plans to execute and how these plans should be interleaved so as to maximise the number of goals achieved and minimise the variance in elapsed time. If the intention selected for execution at the previous cycle posted a subgoal, *SA* explores, through sampled pseudorandom simulation, the implications of intending all relevant applicable plans for the subgoal (and the possible subplans of those plans) and their possible interleavings with all possible ways of achieving the agent's other intentions, for the number of goals achieved and the elapsed time.

As such it assumes responsibility for decisions that some agent platforms traditionally leave to the developer, e.g., the S_O and S_I selection functions of Jason [4] allow a developer to customise Jason's plan and intention selection for a particular application domain. While such customisation may be necessary or desirable for some problems, it requires specialist expertise on part of the developer to program at the level of the interpreter rather than writing BDI agent code.[2] In addition, the developer no longer has to anticipate and control possible interactions between intentions using atomic constructs. On the other hand, *SA* does entail additional developer effort in defining the pre- and postconditions of actions. However alternative approaches to avoiding conflicts between intentions, e.g., SI, require similar information in the form of pre- and postconditions of plans and goals. (Note that in *SA*, pre- and postconditions of plans are derived from the pre- and postconditions of the actions comprising the plan.) Some BDI programming languages already require the developer to specify the pre- and postconditions of actions, e.g., actions in GOAL [13], belief-update actions in 2APL [9]. For languages such as Jason, our experimental results indicate the kinds of scenarios where the additional effort of specifying the pre- and postconditions of actions would be worthwhile in terms of increased agent performance.

We conclude this section by briefly outlining some simple optimisations of our current implementation. As described in Section 4, *SA* builds a search tree at each deliberation cycle. As we show below, in many cases *SA* is able to find an interleaving that achieves all the agent's goals. If the environment does not change, the next step in this interleaving can simply be returned at subsequent deliberation cycles without further computation. It is only necessary to build a new search tree when the environment changes or the agent adopts a new top-level goal. In low dynamism environments, this means the already small overhead of *SA* (see Section 6.3) can be amortised over multiple execution steps.

6. EVALUATING SA

In this section we compare the performance of *SA* with existing approaches to intention selection used in practical implementations of agent programming languages (first in first out, round robin), and from the theoretical literature (summary information, coverage-based). We consider both static and dynamic environments in synthetic domains, and a more realistic scenario based on the Miconic-10 elevator domain [14].

First in first out (FIFO) executes each of the agent's intentions to completion (the goal is achieved or the next step in the intention cannot be executed) before starting to execute the next intention (this is termed a *non-interleaved execution strategy* in [2]). FIFO minimises interactions between intentions, however it has the disadvantage that the achievement of some goals may be significantly delayed compared to other goals.

Round robin (RR) attempts to ensure 'fairness' between intentions by executing a fixed number of steps of each intention in turn. A disadvantage of round robin is that it increases the number of possible conflicts between intentions: the interleaving of steps in different intentions may destroy a precondition established by a step in another intention before the action that requires the precondition is executed.[3] Round robin is the default intention selection function used by Jason [4]; some platforms, such as JACK [30] allow the developer to choose between round robin and first in first out.

In *Summary Information*-based intention selection (SI), summary information about pre-, in- and postconditions is propagated up each goal-plan tree to allow reasoning about interactions between trees. For example, if there are two possible plans for a goal g which both have p as a postcondition, then it is possible to infer

[2] *SA* could be extended to allow customisation of the value function(s) that determine which interleavings are preferred; however our current implementation does not support this.

[3] The same problems occur if the agent executes a step of each intention in parallel at each cycle as in, e.g., 2APL [9].

that p will definitely occur as a result of achieving g, and that any other goal that brings about $\neg p$ may cause a conflict. SI uses this information to protect active p-effects and in-conditions. If adopting a new goal will result in a conflict, i.e., a postcondition of the new goal destroys an active p-effect or in-condition, then progression of the intention containing the new goal is delayed until the p-effect or in-condition is no longer active.

Coverage-based intention selection (C) is based on the notions of *coverage* and *overlap* defined in Section 2. It prioritises the progression of intentions with lowest coverage, as they are more likely to fail due to changes in the environment. Waters et al. [27, 28] implemented and evaluated the approach proposed in [23] (which they call c0), and a preemptive variant of c0 which they call c1. c0 executes the progressable intention with lowest coverage and maintains focus as long as possible. Only when the current intention is finished or blocked will c0 recalculate coverage for all intentions and change focus to the progressable intention with lowest coverage. In c1, whenever a sub-goal is posted, the coverage of each intention is recalculated and its progressibility determined. As a result, c1 will change focus to a lower coverage progressable intention, even if the current intention is progressable.

FIFO does not interleave the execution of intentions. SI and C interleave intentions at the plan level. RR and *SA* interleave intentions at the action level.

FIFO, RR, SI and C implicitly prioritise different criteria. FIFO prioritises minimisation of conflicts at the expense of 'fairness'. Round robin introduces more potential for conflicts, but is more fair. Summary information based and coverage-based intention selection lie somewhere in between. We therefore evaluate the performance of each approach on two criteria: the number of goals achieved, and the variance in elapsed time (see Section 4).[4] In Sections 6.1.1 and 6.1.2, we only report variance results where the number goals achieved is > 8; as the number of intentions which complete successfully decreases, the variance value conflates the ability to achieve goals with the fairness of intention selection. For example, intentions that fail increase the variance of RR and decrease the variance of FIFO.

6.1 Synthetic Domains

In the interests of generality, our first evaluation is based on sets of randomly-generated, synthetic goal-plan trees representing the current intentions of an agent in a simple environment. By controlling the characteristics of the trees, and the dynamism of the environment, we can evaluate the performance of each scheduling algorithm under different conditions.

The environment is defined by a set of propositions that may appear as pre-, in- or postconditions of a goal-plan tree. Each environment variable is modelled as a Poisson process with specified mean, allowing the frequency with which the environment changes the value of the variable to be controlled. For a static environment, the means of all Poisson processes are set to 0. For the experiments reported below, the environment consists of 20 propositions.

Each synthetic goal-plan tree is specified by five parameters: the depth of the goal-plan tree, the plan branching factor (the maximum number of plans that can be used to achieve a goal), the goal branching factor (the maximum number of sub-goals a plan may have), the maximum number of actions in a plan and the number of environment variables that may appear in the tree.

For the experiments reported below, we assume there are at most two plans for each goal, and that each plan has a single environment variable as its precondition. The precondition may either depend on the current state of the environment or be established by the postcondition of a previous step in the goal-plan tree, i.e., by a p-effect. Where the preconditions of the plans for a goal g are not established by p-effects (e.g., top-level goals), a proposition p is chosen randomly from the set of environment variables, and one plan for g has the precondition p and the other plan has precondition $\neg p$. Each plan consists of a number of actions followed by a single subgoal. The first action in the plan has the plan's precondition as its own precondition. The remaining actions either have the plan precondition as their own precondition or their precondition is established by a previous action in the plan. The postcondition of an action is selected randomly from the set of environment variables. The constraints ensure that: (a) each plan is well formed (the plan can be successfully executed in some environment), and (b) taken individually, each goal-plan tree is executable (there is at least one way to achieve the top-level goal in all (static) environments).

These parameters generate trees similar to those used in previous evaluations of SI [21] and C [27, 28]. The main differences are that: unlike [21] goals do not have postconditions and plans and goals do not have in-conditions (for compatibility with [27, 28]); and, unlike [27, 28], environment variables are shared by all goal-plan trees (for compatibility with [21]).

6.1.1 Static Environment

Our first set of experiments evaluated the performance of FIFO, RR, SI and *SA* in a static environment. (C is designed primarily for dynamic environments and is considered in Section 6.1.2.) For each experiment, we generated 50 sets of 10 goal-plan trees using the parameters specified above (depth 5, plan branching factor of at most 2, goal branching factor 1). Other parameters of the trees were varied in each experiment as detailed below. The environment consisted of 20 variables, all of which can be chosen for pre- and postconditions in each tree. *SA* was configured with a dynamism threshold $\delta = 0.1$, and to perform 100 iterations ($\alpha = 100$) and 10 random simulations at each iteration (β = 10).

We first consider two parameters which influence the number of potential conflicts between intentions: the number of plan preconditions established by p-effects and the coverage of the goal plan trees. These parameters were varied to create intention selection scenarios of increasing difficulty.

Experiment 1 In the first experiment we varied the percentage of plan preconditions established by p-effects from 33% to 100%. Goal-plan trees with lower % p-effects have higher coverage (plans whose precondition is not established by a p-effect have have preconditions p and $\neg p$, and hence no overlap), and therefore less potential for conflicts (whatever the value of p the agent has an applicable plan). We set the average p-effect size at level 5 to be 3 (i.e., on average, the precondition of a plan is established by an action executed three steps earlier, in the parent plan), and each plan contained three actions.

% p-effects	FIFO	RR	SI	*SA*
33%	10 (1856)	3.04	10 (291)	10 (17)
66%	10 (1856)	3.02	9.88 (320)	10 (20)
100%	10 (1856)	2.82	9.72 (373)	10 (19)

Table 1: Goals & variance with increasing % p-effects

Table 1 shows the average number of goals achieved and variance in goal achievement time (in brackets, smaller numbers are better) for each scheduling algorithm for each percentage of p-effects considered. (Recall that we only report variance results where the number goals achieved is > 8.) As can be seen, RR is able to achieve only 3.04 goals on average when 33% of plan preconditions are estab-

[4] An alternative definition of fairness is used as an evaluation criterion in [28].

lished by p-effects. This is because, even though 67% of subgoals have 100% coverage, there are still many points at which an action from another intention can destroy the precondition of a plan or an action. With 33% p-effects SI is able to achieve all 10 goals, however as the percentage of p-effects increases, the number of goals achieved decreases slightly. FIFO and *SA* are able to achieve all 10 goals even with 100% p-effects. However the variance in completion time is much higher with FIFO (1856) than with *SA* (20).

Experiment 2 We investigated the effect of reducing coverage. When generating the goal-plan trees, we varied the probability of a goal having only a single plan. 25% 1-plan means that with probability 0.25 only one plan is generated for a goal, and so on. Increasing the probability that a goal has only one plan reduces coverage, and increases the likelihood of conflicts between intentions. The percentage of plan preconditions established by p-effects was 100% (i.e., all plans to achieve subgoals have their preconditions established by preceding actions); other parameters were as in Experiment 1.

% 1-plan	FIFO	RR	SI	*SA*
0%	10 (1856)	2.82	9.72 (373)	10 (19)
25%	8.92 (1603)	2.76	9.24 (455)	10 (22)
50%	7.92	2.44	8.88 (475)	10 (27)
75%	7.36	2.12	8.52 (489)	10 (28)

Table 2: Goals & variance with increasing % 1-plan (100% p-effects)

The results are shown in Table 2. As can be seen, for FIFO, RR and SI, as coverage decreases, the average number of goals achieved also decreases. This effect is particularly marked in the case of FIFO (if a top-level goal has only one plan, the postcondition of an action in a previously executed intention which destroys the precondition of the plan will make the intention unexecutable). *SA* is able to achieve 10 goals at all coverage levels. For *SA* and SI variance increases as coverage decreases. However *SA* has significantly lower variance than SI for all coverage levels, as SI approximates FIFO when protecting p-effects.

Next we investigated the performance of each approach when intentions are more complex and hence the likelihood of negative interactions is greater. In these experiments, we varied the p-effect size, and the number of actions in each plan.

Experiment 3 We investigated the effect of increasing p-effect size. We generated trees in which the precondition of each plan is chosen from environment variables appearing in the postcondition of any preceding action in the tree. For trees of depth 5, this gives an average p-effect size of 5 at level 5 (i.e., on average, the precondition of a plan is established by an action executed five steps earlier, in the parent of the parent of the plan). As in Experiment 2, we varied the probability of a goal having only a single plan from 0% to 75%; all other parameters are the same as in Experiment 2.

% 1-plan	FIFO	RR	SI	*SA*
0%	10 (1856)	2.68	9.32 (374)	10 (19)
25%	9.08 (1617)	2.64	9.04 (460)	10 (22)
50%	7.94	2.2	8.6 (471)	10 (25)
75%	6.94	2	8.34 (490)	10 (27)

Table 3: Goals & variance with increasing % 1-plan (100% p-effects, average p-effect size 5)

The results are shown in Table 3. With a larger p-effect size, FIFO, RR and SI perform slightly worse in terms of goals achieved than in Experiment 2, while *SA* is still able to achieve 10 goals in all cases. The variance for all approaches is essentially the same as in Experiment 2.

Experiment 4 We increased the number of actions in each plan to 5. As in Experiment 2, we varied the probability of a goal having only a single plan from 0% to 75%; all other parameters are the same as in Experiment 2.

% 1-plan	FIFO	RR	SI	*SA*
0%	10 (5156)	1.64	9.28 (544)	10 (40)
25%	8.92 (4564)	1.56	9 (755)	10 (58)
50%	7.36	1.36	8.52 (843)	10 (63)
75%	6.52	1.32	8.3 (874)	10 (66)

Table 4: Goals & variance with increasing % 1-plan (100% p-effects, each plan contains 5 actions)

The results are shown in Table 4. As can be seen, the performance of FIFO, RR and SI decrease when plans have 5 actions, with a more marked decline in the case of RR. SI shows a significant increase in variance. However, *SA* still achieves all 10 goals, and while the variance is increased, it is much smaller than that for the other approaches.

6.1.2 Dynamic Environment

Our second set of experiments evaluated the performance of FIFO, RR, *SA*, and two variants of C from [27, 28], c0 and c1, in a dynamic environment. In the dynamic case, all the Poisson processes controlling the environment variables have the same non-zero mean. (In reality, different environment variables will change value at different rates, however for random trees we believe this is a reasonable approximation.) The larger the mean, the greater the probability an environment variable will change value (from p to $\neg p$ or vice versa). We used three values for the mean: 0.01, which results in an average of 0.2 events (changes in the value of an environment variable) per deliberation cycle, 0.05 (which gives an average of 0.97 events per cycle), and 0.1 (which gives 1.91 events per cycle). In the dynamic case, the environment is first updated to reflect the execution of the next step of the selected intention (as in the static case). The Poisson processes controlling the environment variables are then evaluated, possibly resulting in additional changes to the environment. The resulting environment state is then 'sensed' by the agent at the beginning of the next deliberation cycle. For each experiment, we generated 50 sets of 10 goal plan trees using the same parameters as in the static experiments. As before, $\delta = 0.1$, $\alpha = 100$ and $\beta = 10$.

We first consider the effect of increasing dynamism and reduced coverage.

Experiment 5 In the fifth experiment, we used the same parameters as in first row in Table 2, i.e., all plans have two goals (0% 1-plan), %p-effects was 100%, each plan has 3 actions and the p-effect size at level 5 was 3. The means of Poisson process was varied from 0.01 to 0.1.

Dyn	FIFO	RR	c0	c1	*SA*
0.01	8.92 (1625)	2.24	9.52 (1849)	8.88 (96)	9.82 (95)
0.05	5.40	1.06	8.62 (1830)	8.64 (94)	9.40 (95)
0.10	3.28	1	8.16 (1962)	8.56 (97)	9.08 (95)

Table 5: Goals & variance with increasing dynamism (100% p-effects, 0% 1-plan)

The results are shown in Table 5. As can be seen, for all approaches, as the means of Poisson processes increase, the average number of goals achieved decreases. This effect is particularly marked in the case of FIFO. c0 performs better than c1 when the Poisson mean is small, and c1 has a better performance than c0 when Poisson means are larger. *SA* outperforms the other approaches in terms of

the number of goals achieved. The variance in goal achievement time of *SA* is similar to c1 and significantly lower than c0.

Experiment 6 We set the probability of a goal having only a single plan to be 0.25 (i.e., the second row in Table 2); the means of the Poisson processes varied from 0.01 to 0.1; all other parameters are the same as in Experiment 2.

Dyn	FIFO	RR	c0	c1	*SA*
0.01	7.90	1.82	9.12 (1877)	8.40 (116)	9.74 (114)
0.05	4.74	1.14	7.98	8.26 (118)	9.28 (120)
0.10	2.54	0.72	7.64	8.22 (124)	8.94 (123)

Table 6: Goals & variance with increasing dynamism (100% p-effects, 25% 1-plan)

The results are shown in Table 6. As can be seen, for all approaches and levels of dynamism, with reduced coverage (25% 1-plan) the average number of goals achieved decreases and the variance in completion time increases. However, *SA* still achieves the highest number of goals of any of any approach (between 8%–16% better than c1), and its variance in goal achievement time is comparable to c1.

We then investigated the effect of increasing dynamism when intentions are more complex.

Experiment 7 We increased the p-effect size to 5 at level 5. All other parameters were as in the first row in Table 2, and we again varied the means of Poisson process from 0.01 to 0.1.

Dyn	FIFO	RR	c0	c1	*SA*
0.01	8.88 (1641)	2	9.42 (1863)	8.6 (100)	9.8 (96)
0.05	5.34	1.02	8.48 (1876)	8.54 (108)	9.32(97)
0.10	3.12	0.94	8.02(1978)	8.48 (110)	8.98 (96)

Table 7: Goals & variance with increasing dynamism (100% p-effects, p-effect size 5)

The results are shown in Table 7. Compared to Experiment 5, the performance of c0, c1 decreases as we increase the p-effect size. However, *SA* still outperforms other approaches on the number of goals achieved and its variance in completion time is slightly lower than that of c1.

Experiment 8 We increased the number of actions in each plan to 5; other parameters were same as the first row in Table 2, and the means of the Poisson process were varied from 0.01 to 0.1.

Dyn	FIFO	RR	c0	c1	*SA*
0.01	8.4 (4664)	1.08	9.22 (4605)	8.32 (360)	9.52 (346)
0.05	3.76	0.62	8.18 (4765)	8.26 (387)	9.08(367)
0.10	1.84	0.42	7.88	8.18 (382)	8.72 (386)

Table 8: Goals & variance with increasing dynamism (100% p-effects, each plan contains 5 actions)

The results are shown in Table 8. As can be seen, compared to Experiment 5, the performance of FIFO, RR, c0, c1 and *SA* decreases as we increase the number of actions in each plan. However, *SA* still outperforms other approaches on the number of goals achieved and has a similar variance to c1.

Overall, *SA* outperforms FIFO, RR and SI on the number of goals achieved in static environments. In simpler scenarios, FIFO and SI are able to achieve all 10 goals. However, as the scenarios become more challenging, their performance declines (FIFO more markedly than SI). *SA* also has a significantly lower variance in elapsed time (particularly compared to FIFO), as it allows interleaving at the action level. In dynamic environments, *SA* outperforms FIFO, RR, c0 and c1 in terms of the number of goals achieved. The variance in elapsed time time of *SA* is comparable to that of c1, and significantly lower than that of c0. While the relative performance advantage of *SA* in a dynamic environment is smaller than in a static environment, it consistently outperforms c1 in terms of number of goals achieved by between 5% and 10%.

6.2 Miconic-10 Elevator Domain

In this section we evaluate the performance of *SA* in a more realistic scenario based on the Miconic-10 elevator domain [14].[5] Unlike traditional elevators, the Miconic-10 elevator allows passengers to enter their destination floor when calling the elevator using a 10-digit keypad installed in each elevator lobby. In the scenario, the Miconic-10 elevator is controlled by a BDI agent program. Using standard techniques for translating HTN planning problems into BDI agent programs, e.g., [10], we translated the methods and operators in the Miconic-10 Elevator HTN planning domain given in [14], to goals, plans and actions in a BDI agent program. The goal-plan tree for the resulting BDI program is shown in Figure 1.

In our experiments, there is a single Miconic-10 elevator installed in a 10 floor building. Passengers may move from floor to floor using the elevator. Requests to travel to a particular floor give rise to top-level goals (one for each request) which are achieved by the BDI agent program. For simplicity, we assume there is no limitation on the number of passengers that can be in the elevator at any one time.

We evaluated the performance of *SA* in two scenarios. In the first experiment, all goals (elevator requests) are given at deliberation cycle 0 (corresponding to the static environment in the synthetic domains). We generated 50 sets of 10 goals to travel between two randomly chosen distinct floors. *SA* was configured as in Section 6.1.1, i.e., $\delta = 0.1$, $\alpha = 100$ and $\beta = 10$.

FIFO	RR	SI	*SA*
10 (123)	1.10	10 (33)	10 (28)

Table 9: Goals & variance for 10 goals

The number of goals achieved and variance in elapsed time (in parentheses) for FIFO, RR, SI and *SA* are shown in Table 9. As can be seen, with the exception of RR, all approaches achieve 10 goals. However FIFO has significantly higher variance than SI and *SA*, as FIFO executes the intentions sequentially.

In our second experiment, we use a more realistic model in which elevator requests arrive over time. As before, a total of 10 top-level goals must be achieved. However, in this scenario two goals are given at deliberation cycle 0, and the remaining top-level goals are posted during the progression of the agent's current intentions. We varied $\#g$, the number of new top-level goals generated after executing each primitive action, from 0.33 to 3. When $\#g = 0.33$, a new top-level goal is posted after every 3 primitive actions; when $\#g = 3$, 3 new top-level goals are posted after executing each primitive action. The *SA* search configuration was the same as in the first scenario.

$\#g$	FIFO	RR	c0 & c1	*SA*
0.33	10 (14)	2.16	10 (14)	10 (1)
0.50	10 (36)	1.56	10 (36)	10 (1)
1	10 (71)	1.20	10 (71)	10 (9)
2	10 (95)	1.10	10 (95)	10 (18)
3	10 (102)	1.10	10 (102)	10 (21)

Table 10: Goals & variance with decreasing time between goals

[5] An elevator controller written in GOAL is evaluated in [1].

The results are shown in Table 10. As in the first scenario, with the exception of RR, all approaches achieve 10 goals. However in the dynamic setting, c0 and c1 have the same variance as FIFO (as the *MoveTo* subgoal has 100% coverage, achieving a *MoveTo* subgoal does not change the coverage of the current intention and so both c0 and c1 behave like FIFO). In contrast *SA* has significantly lower variance (i.e., the waiting times for passengers is 'fairer'), as it is able to interleave actions in different intentions without destroying preconditions.

6.3 Computational Overhead of SA

The computational overhead of *SA* depends on the search configuration: how many iterations of the algorithm are performed α and how many random simulations are performed at each node β. With the search configuration used for the experiments above ($\alpha = 100$ and $\beta = 10$), *SA* requires 35 milliseconds to compute a complete interleaving of actions in 10 synthetic goal-plan trees (150 primitive actions). 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 overhead of *SA* is effectively amortised over the execution of all 10 intentions, and the amortised CPU time required to selection an intention is approximately 0.2 milliseconds per deliberation cycle. In a dynamic environment, the interleaving must be recomputed whenever the agent's beliefs change, so the cost of intention selection will be correspondingly higher.

7. RELATED WORK

In addition to the work of Thangarajah et al. [22, 21, 23], Waters et al. [27, 28] and Yao et al. [31] discussed above, a number of other approaches to scheduling intentions to avoid conflicts have been proposed in the literature.

Shaw and Bordini have proposed approaches to intention selection based on Petri nets [19] and constraint logic programming [20]. As in [22, 21, 23, 27, 28, 31]), 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 [11] together with Design-To-Criteria (DTC) scheduling [26] have been used in agent architectures such the Soft Real-Time Agent Architecture [25] and AgentSpeak(XL) [3] 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, DTC was used to schedule execution of AgentSpeak intentions at the level of individual plans [3]. The TÆMS relations between plans required to generate a schedule (*enables*, *facilitates* and *hinders*) were specified as part of the agent program. In contrast *SA* 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 [17] 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 [29], Wilkins et al. present the Cypress architecture which combines the the PRS-CL reactive executor 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 focuses on the generation of new plans to recover from plan failures, rather than interleaving intentions so as to avoid conflicts.

In [32], Yao et al. present an approach to recovering from execution failures in BDI agent programs that relies on progressing executable intentions so as to (re)establish the precondition of the next step in an unexecutable intention as a 'side effect'. As in the work presented here, they interleave intentions at the action level. However the focus of their work is on minimising backtracking (trying an alternative plan when an intention becomes unexecutable), rather than coverage and fairness. Moreover, their approach is based on MCTS (rather than SP-MCTS), and their evaluation is limited to static domains.

There has also been work on avoiding conflicts in a multi-agent setting. For example, Clement and Durfee [6, 7, 8] 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 [12], 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.

8. DISCUSSION & FUTURE WORK

We presented *SA*, an extension of the stochastic scheduling approach proposed in [31], that allows the interleaving of primitive actions in different intentions, and which takes both the dynamism of the environment and fairness into account when choosing which intention to progress. Action level intention selection allows conflicts that cannot be resolved by scheduling at the plan level to be avoided. We evaluated the performance of *SA* in static and dynamic environments, both in synthetic domains and a more realistic scenario based on the Miconic-10 elevator domain. Our results indicate that our approach out-performs RR, FIFO SI, and C, and is capable of successfully scheduling intentions so as to achieve their top-level goals even when the number of potential conflicts between intentions is high. Moreover, in the scenarios considered, the computational overhead of *SA* is modest.

Our approach has a number of limitations. When the agent's beliefs or goals change, our current implementation simply regenerates the SP-MCTS search tree from scratch, rather than merging belief and goal updates into the search tree from the previous deliberation cycle. A direction for future work is therefore to investigate ways of re-using an existing SP-MCTS search tree when the interleaving must be recomputed. Another line of future work is to consider intention selection where top-level goals are associated with deadlines as in, e.g., [24].

Acknowledgments

We would like to thank John Thangarajah for many helpful discussions relating to the work presented here.

REFERENCES

[1] N. Alechina, T. Behrens, M. Dastani, K. Hindriks, J. Hubner, B. Logan, H. Nguyen, and M. van Zee. Multi-cycle query caching in agent programming. In M. desJardins and M. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013)*, volume 1, pages 32–38, Bellevue, Washington, July 2013. AAAI, AAAI Press.

[2] N. Alechina, M. Dastani, B. Logan, and J.-J. C. Meyer. Reasoning about agent deliberation. In G. Brewka and J. Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 16–26, Sydney, Australia, September 2008. AAAI.

[3] R. Bordini, A. L. C. Bazzan, R. d. O. Jannone, D. M. Basso, R. M. Vicari, and V. R. Lesser. 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)*, pages 1294–1302, 2002.

[4] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. Wiley, 2007.

[5] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.

[6] B. J. Clement and E. H. Durfee. Theory for coordinating concurrent hierarchical planning agents using summary information. In J. Hendler and D. Subramanian, editors, *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, pages 495–502, Orlando, Florida, USA, July 1999. AAAI Press / The MIT Press.

[7] B. J. Clement and E. H. Durfee. Performance of coordinating concurrent hierarchical planning agents using summary information. In *Proceedings of the 4th International Conference on Multi-Agent Systems*, pages 373–374, Boston, MA, USA, July 2000. IEEE Computer Society.

[8] B. J. Clement, E. H. Durfee, and A. C. Barrett. Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research*, 28:453–515, 2007.

[9] M. Dastani. 2APL: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.

[10] L. de Silva and L. Padgham. A Comparison of BDI Based Real-Time Reasoning and HTN Based Planning. In G. I. Webb and X. Yu, editors, *AI 2004: Advances in Artificial Intelligence, Proceedings of the 17th Australian Joint Conference on Artificial Intelligence*, volume 3339, pages 1167–1173, Cairns, Australia, December 2004. Springer.

[11] K. S. Decker and V. R. Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 2:215–234, 1993.

[12] E. Ephrati and J. S. Rosenschein. A framework for the interleaving of execution and planning for dynamic tasks by multiple agents. In C. Castelfranchi and J. Müller, editors, *From Reaction to Cognition, Proceedings of the 5th European Workshop on Modelling Autonomous Agents (MAAMAW'93)*, pages 139–153, Neuchatel, Switzerland, August 1993. Springer.

[13] K. V. Hindriks. Programming rational agents in GOAL. In A. El Fallah Seghrouchni, J. Dix, M. Dastani, and R. H. Bordini, editors, *Multi-Agent Programming: Languages, Tools and Applications*, pages 119–157. Springer, 2009.

[14] J. Koehler and K. Schuster. Elevator Control as a Planning Problem. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 331–338, Breckenridge, CO, USA, April 2000. AAAI.

[15] S. Matsumoto, N. Hirosue, K. Itonaga, N. Ueno, and H. Ishii. Monte-Carlo tree search for a reentrant scheduling problem. In *40th International Conference on Computers and Industrial Engineering (CIE)*, pages 1–6, Awaji, 2010. IEEE.

[16] A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pages 439–449, 1992.

[17] S. Sardiña, L. de Silva, and L. Padgham. Hierarchical planning in BDI agent programming languages: a formal approach. In H. Nakashima, M. P. Wellman, G. Weiss, and P. Stone, editors, *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 1001–1008, Hakodate, Japan, May 2006. ACM.

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

[19] P. H. Shaw and R. H. Bordini. Towards alternative approaches to reasoning about goals. In M. Baldoni, T. C. Son, M. B. van Riemsdijk, and M. Winikoff, editors, *Proceedings of the Fifth International Workshop on Declarative Agent Languages and Technologies (DALT 2007)*, volume 4897, pages 104–121, Honolulu, HI, USA, May 2007. Springer.

[20] P. H. Shaw and R. H. Bordini. An alternative approach for reasoning about the goal-plan tree problem. In H. Coelho, R. Studer, and M. Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, volume 215, pages 1035–1036, Lisbon, Portugal, August 2010. IOS Press.

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

[22] J. Thangarajah, L. Padgham, and M. Winikoff. Detecting & avoiding interference between goals in intelligent agents. In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 721–726, Acapulco, Mexico, August 2003. Morgan Kaufmann.

[23] J. Thangarajah, S. Sardina, and L. Padgham. Measuring plan coverage and overlap for agent reasoning. In W. van der Hoek, L. Padgham, V. Conitzer, and M. Winikoff, editors, *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, pages 1049–1056. IFAAMAS, June 2012.

[24] K. Vikhorev, N. Alechina, and B. Logan. Agent programming with priorities and deadlines. In K. Turner, P. Yolum, L. Sonenberg, and P. Stone, editors, *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 397–404, Taipei, Taiwan, May 2011.

[25] R. Vincent, B. Horling, V. Lesser, and T. Wagner. Implementing soft real-time agent control. In *Proceedings of*

the Fifth International Conference on Autonomous Agents (AGENTS'01), pages 355–362, 2001.

[26] T. Wagner, A. Garvey, and V. Lesser. Criteria-directed heuristic task scheduling. *International Journal of Approximate Reasoning*, 19:91–118, 1998.

[27] M. Waters, L. Padgham, and S. Sardina. Evaluating coverage based intention selection. In A. Lomuscio, P. Scerri, A. Bazzan, and M. Huhns, editors, *Proceedings of the 13th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2014)*, pages 957–964. IFAAMAS, 2014.

[28] M. Waters, L. Padgham, and S. Sardiña. Improving domain-independent intention selection in BDI systems. *Autonomous Agents and Multi-Agent Systems*, 29(4):683–717, 2015.

[29] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):121–152, 1995.

[30] M. Winikoff. JACK intelligent agents: An industrial strength platform. In R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, pages 175–193. Springer, 2005.

[31] Y. Yao, B. Logan, and J. Thangarajah. SP-MCTS-based intention scheduling for BDI agents. In T. Schaub, G. Friedrich, and B. O'Sullivan, editors, *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI-2014)*, pages 1133–1134, Prague, Czech Republic, August 2014. ECCAI, IOS Press.

[32] Y. Yao, B. Logan, and J. Thangarajah. 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, February 2016. AAAI, AAAI Press. (In press).