Prottle: A Probabilistic Temporal Planner

Iain Little, Douglas Aberdeen, and **Sylvie Thiébaux**
National ICT Australia & Computer Sciences Laboratory
The Australian National University
Canberra, ACT 0200, Australia

Abstract

Planning with concurrent durative actions and probabilistic effects, or *probabilistic temporal planning*, is a relatively new area of research. The challenge is to replicate the success of modern temporal and probabilistic planners with domains that exhibit an interaction between time and uncertainty. We present a general framework for probabilistic temporal planning in which effects, the time at which they occur, and action durations are all probabilistic. This framework includes a search space that is designed for solving probabilistic temporal planning problems via heuristic search, an algorithm that has been tailored to work with it, and an effective heuristic based on an extension of the planning graph data structure. `Prottle` is a planner that implements this framework, and can solve problems expressed in an extension of PDDL.

Introduction

Many real-world planning problems involve a combination of both time and uncertainty (Bresina *et al.* 2002). For instance, Aberdeen *et al.* (2004) investigate military operations planning problems that feature concurrent durative actions, probabilistic timed effects, resource consumption, and competing cost measures. It is the potential for such practical applications that motivates this research.

Probabilistic temporal planning is the combination of concurrent durative actions and probabilistic effects. This unification of the disparate fields of probabilistic and temporal planning is relatively immature, and presents new challenges in efficiently managing an increased level of expressiveness.

The most general probabilistic temporal planning framework considered in the literature is that of Younes and Simmons (2004). It is expressive enough to model generalised semi-Markov decision processes (GSMDPs), which allow for exogenous events, concurrency, continuous-time, and general delay distributions. This expressiveness comes at a cost: the solution methods proposed in (Younes & Simmons 2004) lack convergence guarantees and significantly depart from the traditional algorithms for both probabilistic and temporal planning. Concurrent Markov decision processes (CoMDPs) are a much less general model that simply allows instantaneous probabilistic actions to execute concurrently (Guestrin, Koller, & Parr 2001; Mausam & Weld 2004). Aberdeen *et al.* (2004) and Mausam and Weld (2005) have extended this model by assigning actions a fixed numeric duration. They solved the resulting probabilistic temporal planning problem by adapting existing MDP algorithms, and have devised heuristics to help manage the exponential blowup of the search space.

We present a general framework for probabilistic temporal planning, in which not only do the (concurrent) durative actions have probabilistic effects, but the action durations and discrete effect times can vary probabilistically as well. According to Mausam and Weld (2005), probabilistic planning under these relaxed assumptions goes significantly beyond their own work. Our approach achieves this level of expressiveness while still maintaining a close alignment with existing work in probabilistic and temporal planning (Smith & Weld 1999; Blum & Langford 1999; Bacchus & Ady 2001; Bonet & Geffner 2003).

We start with a brief description of the framework's probabilistic durative actions, define the search space for our probabilistic temporal planning problem, present a trial-based search algorithm to explore it, and devise an effective heuristic that is based on an extension of the planning graph data structure for probabilistic temporal planning. Each of our framework's components is somewhat independent of the others, and could be adapted for other uses. The framework is implemented in a planner called `Prottle`. We demonstrate `Prottle`'s performance on a number of benchmarks, including a military operations planning problem. This paper is based on the thesis (Little 2004), which we refer to for further details.

Probabilistic Durative Actions

`Prottle`'s input language is the temporal STRIPS fragment of PDDL2.1 (Fox & Long 2003), but extended so that effects can be probabilistic, as in PPDDL (Younes & Littman 2004). We also allow effects to occur at any time within an action's duration. The probabilistic and temporal language constructs interact to allow effect times and action durations to vary probabilistically. For clarity, each probabilistic alternative is given a descriptive label.

Figure 1 shows an example action that represents a person jumping out of a plane with a parachute. After 5 units of time, the person makes an attempt to open the parachute. The case where this is successful has the label `parachute-opened`, and will occur 90% of the time; the person will gently glide to safety, eventually landing at time

```
(:durative-action jump
    :parameters (?p - person ?c - parachute)
    :condition (and (at start (and (alive ?p)
                                   (on ?p plane)
                                   (flying plane)
                                   (wearing ?p ?c)))
                    (over all (wearing ?p ?c)))
    :effect (and (at start (not (on ?p plane)))
                 (at end (on ?p ground))
                 (at 5
                   (probabilistic
                     (parachute-opened 0.9 (at 42 (standing ?p)))
                     (parachute-failed 0.1
                       (at 13 (probabilistic
                                 (soft-landing 0.1
                                   (at 14 (bruised ?p)))
                                 (hard-landing 0.9
                                   (at 14 (not (alive ?p)))))))))))
```

Figure 1: An example of an action to jump out of a plane.

42. However, if the parachute fails to open, then the person's survival becomes dependent on where they land. The landing site is apparent at time 13, with a 10% chance of it being soft enough for the person to survive. Alive or dead, the person then lands at time 14, 28 units of time sooner than if the parachute had opened. But regardless of the outcome, or how long it takes to achieve, the action ends with the person's body on the ground.

We treat the structure of an action's possible outcomes as a decision tree, where each non-leaf node corresponds to a probabilistic event, and each leaf node to a possible outcome. As each event is associated with a delay, this structure allows for partial knowledge of an action's actual outcome by gradually traversing the decision tree as time progresses. The duration of an action is normally inferred from the effects that have a numeric time, and depends on the path taken through the decision tree. However, it can be specified absolutely using the `:duration` clause from PDDL2.1.

The decision tree representation assumes that probabilistic alternatives occur to the exclusion of the others. Nevertheless, independent `probabilistic` events are allowed by the input language; any independence is compiled away by enumerating the possibilities.

Search Space

There is a well-established tradition of using the Markov decision process framework to formalise the search space for probabilistic planning algorithms. We take a slightly different approach, by formalising the search space in terms of an AND/OR graph that more closely aligns with the structure of the problem.

An AND/OR graph contains two different types of nodes. In the interpretation that we use, an AND node represents a *chance*, and an OR node a *choice*. We associate choice nodes with the selection of actions, and chance nodes with the probabilistic event alternatives.

Each node is used in one of two different ways: for *selection* or *advancement*. This is similar to what some temporal planners do, where states are partitioned between those that represent action selection, and those that represent time advancement (Bacchus & Ady 2001). This sort of optimisation allows forward-chaining planners to be better guided by heuristics, as action sets are structured into linear sequences.

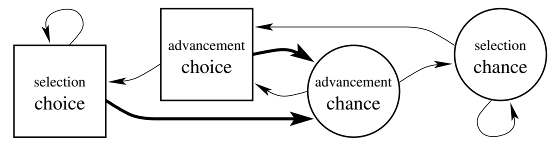

Figure 2: A state machine for valid node orderings. Time may only increase when traversing bold face arcs.

The rules for node succession are defined by Figure 2. They can be summarised as: every successor of a node must either be a selection node of the same type, or an advancement node of the opposite type. Our choice of a search space structure is intended to be used with a 'phased' search, where action selection and outcome determination are kept separate. It might seem that it would be more efficient to have only a single selection phase, where an action's probabilistic branching is dealt with immediately after it is selected, but consider what this does to the problem: we would be assuming that an action's outcome is known as soon as the action starts execution. In contrast, the phased approach allows the time at which this knowledge is available to be accurately represented, by deferring the branching until the appropriate time. This issue of knowledge becomes relevant when concurrency is combined with probabilistic effects. The conservative assumption — that we wait until actions terminate — breaks down when an action's duration can depend on its outcome.

As an example, we now describe a path through such an AND/OR graph, starting from an advancement choice node. First, we choose to start an instance of the `jump` action from Figure 1, which progresses us to a selection choice node. We can now choose either to start another action, or to 'advance' to the next phase; we choose to advance, and progress to an advancement chance node. There is a current probabilistic event with alternatives `parachute-opened` and `parachute-failed`. Let us say that the parachute fails to open for our chosen path, which leaves us at a selection chance node. There are no more events for the current time, so we progress to another advancement choice node. Rather than start another action, we then choose to advance again. The next probabilistic event has alternatives `soft-landing` and `hard-landing`. Let us be nice and say that the person lands on something soft.

Using the graph structure that we have established, we define a state of the search space as a node in an AND/OR graph that is identified by a *time*, *model* and *event queue*. The time of a state is generally the same as its predecessors, but may increase when advancing from choice to chance (see Figure 2). The model is the set of truth values for each of the propositions, and the event queue is a time-ordered list of pending events. An event can be an effect e.g. `(on ?p ground)`, a probabilistic event, or an action execution condition that needs to be checked. When the time is increased, it is to the next time for which an event has been queued. We define the *initial state* as an advancement choice state with time 0, the initial model, and an empty event queue. This reflects the presumption that action selection is the first step in any plan, and that there are no pre-planned events. We define a *goal state* as any state in which the model satisfies the problem's goal.

Heuristic search algorithms associate each state with a value, which is generally either a lower or upper bound on the long-term cost of selecting that state. We associate states with both lower and upper cost bounds. As the search space is explored, the lower bounds will monotonically increase, the upper bounds monotonically decrease, and the actual cost is sandwiched within an ever-narrowing interval. The most important reason for using both bounds is to facilitate a convergence test, although it also introduces the possibility for other optimisations. We say that a state's cost has *converged* when, for a given $\epsilon \geq 0$: $U(s) - L(s) \leq \epsilon$ where U is the upper bound and L the lower bound of state s. For convenience, we restrict costs to the interval $[0, 1]$. The cost of a state is just the probability of the goal being **unreachable** from it if only optimal choices are made. New states are either given a lower bound of 0 and an upper bound of 1, or values that are computed using appropriate heuristic functions (see Heuristics section). Although we only consider probability costs in this paper, the cost scheme that we describe can easily be generalised to include other metrics, such as makespan. The main restriction is that the cost function needs to be bounded.

A state's cost bounds are updated by comparing its current values with those of its successors. We use the following formulae for updating probability costs, where (1)–(2) are for choice states, and (3)–(4) are for chance states:

$$L_{\textbf{choice}}(s) \quad := \quad \max(L(s), \min_{s' \in S(s)} L(s')), \tag{1}$$

$$U_{\textbf{choice}}(s) \quad := \quad \min(U(s), \min_{s' \in S(s)} U(s')), \tag{2}$$

$$L_{\textbf{chance}}(s) \quad := \quad \max(L(s), \sum_{s' \in S(s)} P(s')\, L(s')), \tag{3}$$

$$U_{\textbf{chance}}(s) \quad := \quad \min(U(s), \sum_{s' \in S(s)} P(s')\, U(s')), \tag{4}$$

where S is the set of successors of state s, and P is the probability of s. We define the probability of a selection chance state as the probability of its probabilistic event alternative. The probability of all other states is 1.

In addition to a cost, we also associate each state with a label of either *solved* or *unsolved*. A state is labelled as solved once the benefit of further exploration is considered negligible; for instance, once its cost has converged for a sufficiently small ϵ. The search algorithm is expected to ignore a state once it has been labelled as solved, and to confine its exploration to the remaining unsolved states.

Now that we have established what a state is, we refine the node ordering constraints to restrict the allowable plan structures. The additional rules are:

1. On any path through the search space, an action instance can be started at most once for any particular state time. This rule is a constraint on selection choice states.
2. Every path through a sequence of chance states must represent exactly one alternative from each of the current probabilistic events. This rule is a constraint on selection chance states.

These rules can be easily and efficiently implemented, as shown in (Little 2004).

For an action to be selected, we require that its preconditions are satisfied by the model, and that its start effects are consistent with the other actions that are to be started at the same time. We consider an inconsistency to arise if: (1) a start effect of one action deletes a precondition of another, or (2) both positive and negative truth values are asserted for the same proposition by different start effects. As it is possible for a probabilistic event to occur at the start of an action, we restrict these rules to apply only to start effects that occur irrespective of the outcome.

The selection rules ensure that preconditions are honoured and that a degree of resource exclusion is maintained, but they do not consider other types of conditions or non-start effects. This is deliberate, as with probabilistic outcomes we may not even know whether or not an inconsistency will actually arise. We contend that allowing plans that might not execute successfully in all cases can be preferable to not finding a solution at all. It is then up to the planner to determine whether or not the risk of creating an inconsistency is worth it. For this purpose, there is an inconsistency when: (1) an asserted condition is not satisfied, or (2) both positive and negative truth values are asserted for the same proposition in the same time step. When such an inconsistency is detected, we consider the current state to be a *failure state*; a solved state with a cost of 1.

Search Algorithm

Even though we have not formalised the search space as an MDP, search algorithms that have been designed to solve MDPs can still be applied. Recent probabilistic temporal planners (Aberdeen, Thiébaux, & Zhang 2004; Mausam & Weld 2005) have favoured variants of RTDP, such as LRTDP (Bonet & Geffner 2003). The trial-based derivatives of RTDP are well-suited to probabilistic temporal planning, as they are able to decide between contingency plans without necessarily needing to fully explore all of the contingencies.

The search algorithm that we present is set in a trial-based framework; it explores the search space by performing repeated depth-first probes starting from the initial state, as shown in Figure 3. Search trials are repeatedly performed until the initial state is solved. When a state is first selected, then its successors are created and initialised. A search trial will start backing up cost bounds when it encounters a state that was solved during its initialisation. This can happen if the state is inconsistent, the heuristic cost estimates have been good enough for the state's cost to converge, the state is a goal state, or if the state does not have any successors.

This algorithm was designed to work with a deterministic successor selection function, although it could easily be made probabilistic. We use a function that selects a successor that minimises $P(s)\, U(s)$, and uses $P(s)\, L(s)$ to break ties. The probability ewights only affect the selection of chance states, and focus the search on the most likely alternatives first. Observe that $P(s)\, U(s) = 1$ for choice states until at least one way of reaching the goal has been proved to exist. Prior to this, the lower bound guides the selection

```
SEARCH(initial-state):
repeat
    SEARCH-TRIAL(initial-state)
until LABEL(initial-state) = SOLVED

SEARCH-TRIAL(state):
if LABEL(state) ≠ SOLVED then
    if ¬EXPANDED?(state) then
        EXPAND(state)
    SEARCH-TRIAL(SELECT-SUCCESSOR (state))
    UPDATE-COST-BOUNDS(state)
    UPDATE-LABEL(state)
```

Figure 3: The search algorithm.

of choice states; after, the precedence of cost upper bounds will cause the search to robustify the known solutions.

In part because the selection function is deterministic, it is necessary to limit the search depth to ensure termination. We impose a finite horizon on the search: any state with a time greater than a specified limit is a failure state.[1]

Once the search has terminated, then a solution can be extracted from the search space. We do this by performing a search of all probabilistic alternatives while selecting only optimal choice states with respect to the successor selection function. Because of the labelling scheme, it is possible for this expansion to encounter unsolved states. Another search is started for each such unsolved state. It is possible to use the additional cost information that is produced to improve the solution, which can lessen the effect of choosing $\epsilon > 0$ for convergence. This is described in more detail in (Little 2004). If all heuristics are admissible and $\epsilon = 0$, then only optimal solutions are produced.

This algorithm works with an acyclic search space, where only one state is created for any combination of time, model and event queue. It could also work with a cyclic search space, which might arise when states are disassociated from the absolute timeline, as in (Mausam & Weld 2005).

There is a simple and effective optimisation that we apply to the search algorithm. Observe that the path chosen by the successor selection function is entirely determined by the cost bounds of those states. Furthermore, observe that once a backup fails to alter the current values of a state, no further changes can be made by that backup. As a result, the next search trial will necessarily retrace the unaffected portion of the previous trial. The optimisation is simply to start a new search trial from a state whenever a backup fails to modify its values, and to defer the remainder of the backup until something does change.

Heuristics

Due to the added complexity from combining probabilistic effects and durative actions, effective heuristics are even more critical for probabilistic temporal planning than for simpler planning problems. A popular technique for generating cost heuristics is to use a derivative of the planning graph data structure (Blum & Furst 1997). This has been previously used for both probabilistic (Blum & Langford 1999) and temporal planning (Smith & Weld 1999; Do & Kambhampati 2002), but not for the combination of the two. We extend the planning graph for probabilistic temporal planning and use it to compute an initial *lower bound* estimate for the cost of each newly created state.

The traditional planning graph consists of alternate levels of proposition and action nodes, with egdes linking nodes in adjacent levels. We also include levels of *outcome* nodes, where each such node directly corresponds to a node in an action's decision tree. With this addition, edges link the nodes: proposition to action, action to outcome, outcome to outcome, and outcome to proposition. To cope with the temporal properties, we associate each level with the time step that it represents on the absolute timeline. Excepting persistence actions, we also break the assumption that edges can only link nodes in adjacent levels; all edges involving outcome nodes link levels of the appropriate times. This method of extending the planning graph for temporal planning is equivalent to the *Temporal Planning Graph* (Smith & Weld 1999), in the sense that expanding a level-less graph will traverse the same structure that is represented by an equivalent levelled graph. We find it more convenient to present the planning graph for probabilistic temporal planning in a level-based context.

Generating a planning graph requires a state in order to determine which proposition nodes to include in the initial level. We only generate a graph for the initial state of the problem, although generating additional graphs for other states can improve the cost estimates.[2] The graph expansion continues until the search horizon is reached. This is essential for this heuristic to be admissible, as we need to account for all possible contingencies.

Once the graph is generated, we then assign a vector of costs to each of the graph's nodes. Each component of these vectors is associated with a goal proposition; the value of a particular component reflects the node's ability to contribute to the achievement of the respective goal proposition within the search horizon.[3] In line with our use of costs rather than utilities, a value of 0 means that the node (possibly in combination with others), is able to make the goal inevitable, and a value of 1 means that the node is irrelevant. Cost vectors are first assigned to the nodes in the graph's final level; goal propositions have a value of 0 for their own cost component, and 1 for the others. All other propositions have a value of 1 for all cost components.

Component values are then propagated backwards through the graph in such a way that each value is a lower bound on the actual cost. The specific formulae for cost

[1] We note that if an already solved problem were to be attempted again with a greater search horizon, that much of the work done with the previous horizon can be reused. The cost lower bounds must be discarded, and most solved states need to be reconsidered, but the state structure and cost upper bounds are still valid.

[2] When graphs are generated for non-initial states, the event queue also needs to be accounted for when inferring levels, see (Little 2004) for details on how this is done.

[3] The cost vectors can be extended to include other components, such as for resource usage or makespan.

propagation are:[4]

$$C_o(n,i) \quad := \quad \prod_{n' \in S(n)} C_{p,o}(n',i), \tag{5}$$

$$C_a(n,i) \quad := \quad \sum_{n' \in S(n)} P(n')\, C_o(n',i), \tag{6}$$

$$C_p(n,i) \quad := \quad \prod_{n' \in S(n)} C_a(n',i), \tag{7}$$

where C is the i'th cost component of node n, S are the successors of n, and P is the probability of n. Subscripts are given to C according to node type: o for outcome, a for action and p for proposition. Both C_o and C_p are admissible, and C_a is an exact computation of cost. The products in (5) and (7) are required because it might be possible to concurrently achieve the same result through different means. For example, there is a greater chance of winning a lottery with multiple tickets, rather than just the one. When a planning domain does not allow this form of concurrency, then we can strengthen the propagation formulae without sacrificing admissibility by replacing each product with a $\min$. This effectively leaves us with what has been called *max-propagation* (Do & Kambhampati 2002), which is admissible for temporal planning. Its admissibility in the general case is lost when probabilistic effects are combined with concurrency.

We now explain how the cost vectors are used when computing the lower bound cost estimate for a state generated by the search. This computation involves determining the nodes in the graph that are *relevant* to the state, and then combining their cost vectors to produce the actual cost estimate. When identifying relevant nodes, we need to account for both the state's model and event queue. Accounting for the model is not as simple as taking the corresponding proposition nodes for the current time step. Although this would be admissible, the resulting cost estimates would not help to guide the search; the same estimate would be generated for successive selection states, with nothing to distinguish between them. The way that we actually account for the state's model is to treat as relevant: (1) the nodes from the *next* time step that correspond to current propositions,[5] (2) the nodes for the startable actions that the search algorithm has not already considered for the current time step, if the state is a choice state, and (3) the outcome nodes for the current unprocessed probabilistic events if the state is a chance state. Those proposition or outcome nodes that are associated with an event from the event queue are also considered relevant. This accounts for known future events.

The first step in combining the cost vectors of the relevant nodes is to aggregate each component individually. That is, to multiply — or minimise, if 'max'-propagation is being used — the values for each of the goal propositions to produce a single vector of component values. Then, the actual cost estimate is the maximum value in this vector; the value associated with the 'hardest' goal proposition to achieve.

[4]These formulae assume that every action has at least one precondition; a fake proposition should be used as a dummy precondition if this is not the case.

[5]Or from the current time step if it is also the last; the lack of distinction really does not matter at this point.

Planning graphs usually include mutexes, to represent binary mutual exclusion relationships between the different nodes. Mutexes can be used so that the structure of a planning graph is a more accurate relaxation of the state space. For instance, we know that it is definitely not possible to start an action in a particular time step if there are still mutex relationships between any of its preconditions. For our modified planning graph, we compute mutexes for all of proposition, action and outcome nodes. Of note, all of the usual mutex conditions are accounted for in some way, and there is a special rule for mutexes between action nodes in different levels to account for the temporal dimension. The complete set of mutex rules is described in (Little 2004).

Experimental Results

`Prottle` is implemented in Common Lisp, and is compiled using `CMUCL` version 19a. These experiments were run on a machine with a 3.2 GHz Intel processor and 2 GB of RAM. `Prottle` implements the framework we have described, but two points of distinction should be noted: (1) the search space is acyclic, but only advancement chance states are tested for equivalence, and (2) the planning graph heuristic is implemented using the 'max'-propagation formulae.

We show experimental results for four different problems: `assault-island` (AI), `machine-shop` (MS), `maze` (MZ) and `teleport` (TP). The `assault-island` problem is extracted from a real-world application in military operations planning. It is the same as that used by Aberdeen (2004), but without any metric resource constraints. The `machine-shop` problem is adapted from that used by (Mausam & Weld 2005). Both the `maze` and `teleport` domains were written specifically for `Prottle` and are given in (Little 2004). All results are reported in Table 1. For each problem we vary ϵ and the use of the planning graph heuristic, while recording execution time, solution cost, and the number of states expanded; **time1**, **cost1** and **states1** are for the case where the heuristic is *not* used, and **time2**, **cost2** and **states2** are for when it is. All times are given in seconds. Recall that the costs are probabilities of failure.

The `assault-island` scenario is a military operations planning problem that has 20 different action instances, each of which can be started once, has a duration ranging from 2 to 120, and will succeed or fail with some probability. There is redundancy in the actions, in that individual objectives can be achieved through different means. Results for this problem are presented for two different search horizons: 100 and 120. The values chosen for ϵ are the smallest multiples of 0.1 for which we are able to solve the problem without creating enough states to run out of memory. A recent update to the planner presented in (Aberdeen, Thiébaux, & Zhang 2004) produces solutions of similar quality; with a horizon of 1000 and a small ϵ, it produced a solution that has a failure probability of 0.24. We were only able to solve this problem when the planning graph heuristic was being used.

In `machine-shop`, machines can apply multiple tasks to an object (paint, shape, polish, *etc*) by machines that run in parallel. As with `assault-island` there are 20 action instances. In our version, some actions have nested probabilistic events and outcome-dependant durations. All tests

problem	horizon	ϵ	time1	time2	cost1	cost2	states1	states2
AI	100	0.3	-	103	-	0.344	-	346,100
AI	120	0.6	-	404	-	0.222	-	1,319,229
MS	15	0.0	-	272	-	0.027	-	496,096
MS	15	0.1	-	171	-	0.114	-	309,826
MS	15	0.2	2,431	21	0.119	0.278	13,627,753	6,759
MS	15	0.3	367	235	0.278	0.278	1,950,134	434,772
MZ	10	0.0	195	10	0.178	0.178	1,374,541	13,037
MZ	10	0.1	185	2	0.193	0.178	1,246,159	2,419
MZ	10	0.2	64	1	0.197	0.193	436,876	669
MZ	10	0.3	62	2	0.202	0.193	414,414	1,812
TP	20	0.0	442	< 1	0.798	0.798	3,565,698	3,676
TP	20	0.1	456	< 1	0.798	0.798	3,628,300	2,055
TP	20	0.2	465	< 1	0.798	0.798	3,672,348	2,068
TP	20	0.3	464	< 1	1.000	0.798	3,626,404	1,256

Table 1: Experimental results.

use a horizon of 15. The results are not comparable with those presented in (Mausam & Weld 2005), both because of the changes we made to the problem and the different cost metric (makespan) considered by Mausam and Weld.

The `maze` problem is based on the idea of moving between connected rooms, and finding the keys needed to unlock closed doors. Each of the problem's actions has a duration of 1 or 2, and many of their effects include nested probabilistic events. We used a problem with 165 action instances, although with a much higher degree of mutual exclusion than `assault-island` or `machine-shop`. All tests for this problem have a horizon of 10.

Finally, the `teleport` problem is based on the idea of teleporting between different locations, but only after the teleporters have been successfully 'linked' to their destination. The problem has actions with outcome-dependent durations, and it is possible to opt for slower movement with a higher probability of success. The chosen problem has 63 action instances. All tests have a horizon of 20.

These experiments show that the planning graph heuristic is effective, and can dramatically reduce the number of states that are explored, up to three orders of magnitude. We have hopes that further refinement can make this gap even greater.

The results also show that the choice of ϵ is a control that can affect both search time and solution quality: while ideally we would like to solve all problems with an ϵ of 0, it may be possibly to solve otherwise intractable problems by choosing a higher value. The effect of increasing ϵ is sometimes counter-intuitive, and warrants further explanation. Specifically, sometimes increasing ϵ solves the problem faster, and sometimes it slows the search down; sometimes the increase reduces the quality of the solution, and other times it does not. Increasing ϵ can hinder the search because a trade-off between depth and breadth is being made: by restricting the attention given to any particular state, a greater number of alternatives may need to be considered for the cost bounds of a state's predecessors to converge. And when $\epsilon + C(s) \geq 1$, it is possible for convergence to occur before any way of reaching the goal is found, as demonstrated when $\epsilon = 0.3$ and $C(s) = 0.798$ for TP.

Future Work

Our work on probabilistic temporal planning is in some ways orthogonal to that described in (Mausam & Weld 2004; 2005); we believe that some of the techniques and heuristics that Mausam and Weld describe, such as combo-elimination, eager effects, and hybridization could be adapted for `Prottle`'s framework.

At the moment, the bottleneck that prevents `Prottle` from being applied to larger problems is memory usage. One way this could be improved is to compress the state space as it gets expanded, by combining states of like type and time. Further improvements to the heuristic could also help, such as adapting it to also compute upper bound cost estimates.

There are many ways in which this framework could be made more expressive. The most important practical extensions would be to add support for metric resources, and to generalise costs to support makespan and other metrics. We understand how to do this, but have not yet implemented it.

Acknowledgements

This work was supported by National ICT Australia (NICTA) and the Australian Defence Science and Technology Organisation (DSTO) in the framework of the joint Dynamic Planning Optimisation and Learning Project (DPOLP). NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

References

Aberdeen, D., Thiébaux, S., and Zhang, L. 2004. Decision-theoretic military operations planning. In *Proc. ICAPS*.

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency: A forward chaining approach. In *Proc. IJCAI*.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:281–300.

Blum, A., and Langford, J. 1999. Probabilistic planning in the Graphplan framework. In *Proc. ECP*.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proc. ICAPS*.

Bresina, J.; Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proc. UAI*.

Do, M., and Kambhampati, S. 2002. Planning graph-based heuristics for cost-sensitive temporal planning. In *Proc. AIPS*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Guestrin, C., Koller, D., and Parr, R. 2001. Multiagent planning with factored MDPs. In *Proc. NIPS*.

Little, I. 2004. Probabilistic temporal planning. Honours Thesis, Department of Computer Science, The Australian National University.

Mausam, and Weld, D. 2004. Solving concurrent Markov decision processes. In *Proc. AAAI*.

Mausam, and Weld, D. 2005. Concurrent probabilistic temporal planning. In *Proc. ICAPS*.

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. IJCAI*.

Younes, H. L. S., and Littman, M. 2004. PPDDL1.0: The language for the probabilistic part of IPC-4. In *Proc. International Planning Competition*.

Younes, H. L. S., and Simmons, R. G. 2004. Policy generation for continuous-time stochastic domains with concurrency. In *Proc. ICAPS*.