# Towards Efficient Probabilistic Temporal Planning

**Iain little**
National ICT Australia & Computer Sciences Laboratory
The Australian National University
Canberra, ACT 0200, Australia
**A related paper appears in ICAPS'06.**

## 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 combines concurrent durative actions with 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. Some of our techniques for solving probabilistic temporal planning problems could be applied beyond the context they were developed in, and may prove useful in efficiently solving the simpler subproblems.

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.

The ultimate goal of this research is to produce planners that are expressive enough to support: concurrent durative actions, probabilistic effects, metric resources, and cost functions; while being efficient enough to solve interesting-sized (real-world) problems.

We currently have two separate avenues of research with the aim of achieving this goal. The first approach is to combine a forward-chaining search with effective heuristics. We have developed a probabilistic temporal planner called `Prottle` using this approach (Little, Aberdeen, & Thiébaux 2005). `Prottle` uses a (deterministic) trial-based search algorithm with a heuristic that is based on an extension of the planning graph data structure.

Another approach to planning is the `Graphplan` framework (Blum & Furst 1997). While `Prottle` makes use of the planning graph—a data structure that originates from this framework—it does not use the framework's other key features; in particular, `Prottle` does not use a backward search. The `Graphplan` framework has previously been successfully applied to temporal planning (concurrent durative actions) (Smith & Weld 1999), but had not been successfully applied to probabilistic planning (actions with probabilistic effects) in its entirety. Extensions of this framework for probabilistic planning had been developed (Blum & Langford 1999), but either dispense with the techniques that enable concurrency to be efficiently managed, or are unable to produce optimal contingency plans.

As a way of investigating approaches to compressing the search space for probabilistic temporal planning, our other avenue of research has the goal of implementing a probabilistic temporal planning in the `Graphplan` framework. As the issues relating to probabilistic planning had not been adequately solved, and as a way of managing the complexity, we started by developing a (concurrent) probabilistic planner (Little & Thiébaux 2006). `Paragraph`, the resulting planner, is competitive with the state of the art, producing acyclic or cyclic plans that optionally exploit a problem's potential for concurrency. We are confident that this approach can be extended to the probabilistic temporal context.

This paper gives a brief overview of both `Prottle` and `Paragraph`, and concludes with remarks about our future research intentions. For more detailed descriptions and experimental results, please refer to the respective papers (Little, Aberdeen, & Thiébaux 2005; Little & Thiébaux 2006).

## Prottle

`Prottle` is a probabilistic temporal planner that allows effects, the time at which they occur, and action durations to all be probabilistically determined. Its 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.

```
(: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.

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 $42$. 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.

`Prottle`'s search space is defined in terms of an AND/OR graph. 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.

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

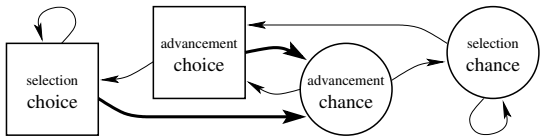


Figure 2: A state machine for valid node orderings. Time may only increase when traversing bold face arcs.

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.

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 scheduled event time.

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. 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 $s$. A state's cost bounds are initially determined using a planning graph-based heuristic, and are updated by comparing its current values with those of its successors.

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 $\epsilon$. The search algorithm ignores a state once it has been labelled as solved, and confines its exploration to the remaining unsolved states.

`Prottle` uses a search algorithm that combines a deterministic search with the convergence and labelling optimisations used by LRTDP (Bonet & Geffner 2003). As with previous probabilistic temporal planners (Aberdeen, Thiébaux, & Zhang 2004; Mausam & Weld 2005), this algorithm is trial-based, and explores the search space by performing repeated depth-first probes starting from the initial state.

## Paragraph

`Paragraph` is a probabilistic planner that finds contingency plans that maximise the probability of reaching the goal within a given time horizon. These solutions are optimal in the non-concurrent case, and optimal for a re-

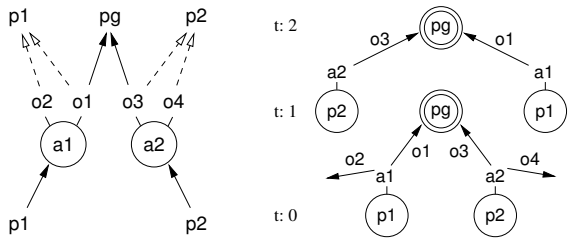


Figure 3: An action-outcome-proposition dependency graph and search space for an example problem.

stricted model of concurrency. A detailed description of this model—and of `Paragraph` in general—is given in (Little, Aberdeen, & Thiébaux 2005).

`Paragraph` extends the `Graphplan` framework to the probabilistic setting. To do this, it is necessary to extend the planning graph data structure to account for uncertainty. We do this by introducing a node for each of an action's possible outcomes, so that there are three different types of nodes in the graph: proposition, action, and outcome. Action nodes are then linked to their respective outcome nodes, and edges representing effects link outcome nodes to proposition nodes. Each persistence action has a single outcome with a single add effect. We refer to a persistence action's outcome as a *persistence outcome*. This extension is functionally equivalent to that described in (Blum & Langford 1999), except that we also adapt the planning graph's mutex propagation rules from the deterministic setting.

The solution extraction step of the `Graphplan` algorithm relies on a backward search through the structure of the planning graph. In classical planning, the goal is to find a subset of action nodes for each level such that the respective sequence of action sets constitutes a valid trajectory. The search starts from the final level of the graph, and attempts to extend partial trajectories one level at a time until a solution is found.

`Paragraph` uses this type of goal-regression search with an explicit representation of the expanded search space. This search is applied exhaustively, to find all trajectories that the `Graphplan` algorithm can find. An optimal contingency plan is formed by linking these trajectories together. This requires some additional computation, and involves using forward simulation through the search space to compute the possible world states at reachable search nodes.

As observed by Blum and Langford (1999), the difficulty with combining probabilistic planning with `Graphplan`-style regression is in correctly and efficiently combining the trajectories. Sometimes the trajectories will 'naturally' join together during the regression, which happens when search nodes share a predecessor through different 'joint outcomes' (sets of outcomes) of the same action set.

Unfortunately, the natural joins are not sufficient to find all contingencies. Consider the problem shown in Figure 3, which we define as:[1] the propositions $p1$, $p2$ and $pg$; $s_0 = \{p1, p2\}$; $G = \{pg\}$; the actions $a1$ and $a2$; and the outcomes $o1$ to $o4$. $a1$ has precondition $p1$ and outcomes $\{o1, o2\}$; $a2$ has precondition $p2$ and outcomes $\{o3, o4\}$. Both actions always delete their precondition; $o1$ and $o3$ both add $pg$. To simplify the example, we prohibit $a1$ and $a2$ from executing concurrently. The optimal plan for this example is to execute one of the actions; if the first action does not achieve the goal, then the other action is executed.

The backward search will correctly recognise that executing $a1$–$o1$ or $a2$–$o3$ will achieve the goal, but it fails to realise that $a1$–$o2, a2$–$o3$ and $a2$–$o4, a1$–$o1$ are also valid trajectories. The longer trajectories are not discovered because they contain a 'redundant' first step; there is no way of relating the effect of $o2$ and the precondition of $a2$, or the effect of $o4$ with the precondition of $a1$. While these undiscovered trajectories are not the most desirable execution sequences, they are necessary for an optimal contingency plan. In classical planning, it is actually a good thing that trajectories with this type of redundancy cannot be discovered, as redundant steps only hinder the search for a single shortest trajectory. Identifying the missing trajectories requires some additional computation beyond the goal regression search. We refer to trajectories that can be found using unadorned goal regression as *natural trajectories*.

The solution we have developed is based on constructing all 'non-redundant' contingency plans by linking together the trajectories that goal regression is able to find. This is sufficient to find an optimal solution, as there always exists at least one non-redundant optimal plan. `Paragraph` combines pairs of trajectories by linking a node in one trajectory to a node in the other. This can be done when a possible world state of the earlier node has a resulting world state that subsumes the goal set of the later node.

A detailed description of `Paragraph`'s acyclic search algorithm follows.[2] The first step is to generate a planning graph from the problem specification. This graph is expanded until all goal propositions are present and not mutex with each other, or until the graph levels off to prove that no solution exists. Assuming the former case, a depth-first goal regression search is performed from a goal node for the graph's final level. This search exhaustively finds all natural trajectories from the initial conditions to the goal. Once this search has completed, the possible world states for each trajectory node are computed by forward-propagation from time 0, and the node/state costs are updated by backward-propagation from the goal node. Potential trajectory joins are detected each time a new node is encountered during the backward search, and each time a new world state is computed during the forward state propagation. Unless a termination condition has been met, the planning graph is then expanded by a single level, and the backward search is performed from a new goal node that is added to the existing search space. This alternation between backward search, state simulation, cost propagation, and graph expansion continues until a termination condition is met. An optimal con-

[1] This problem was used by Blum and Langford (1999) to illustrate the difficulty of using goal-regression for probabilistic planning, and to explain their preference of a forward search in `PGraphplan`.

[2] We have another algorithm for extracting cyclic solutions.

| Horizon | PRTTL Time | NA-PG Time | Cost |
|---|---|---|---|
| 10 | 14.0 | 0.23 | 0.728 |
| 15 | 21.6 | 0.73 | 0.607 |
| 20 | 25.1 | 12.5 | 0.486 |
| 25 | 36.0 | 52.2 | 0.429 |
| 30 | 40.6 | 103 | 0.429 |

(a) g-tire

| Horizon | PRTTL Time | CA-PG Time | PRTTL Cost | CA-PG Cost |
|---|---|---|---|---|
| 5 | 4.38 | 0.08 | 0.272 | 0.204 |
| 6 | 14.9 | 0.13 | 0.204 | 0.193 |
| 7 | 168 | 0.26 | 0.178 | 0.156 |
| 8 | 554 | 0.71 | 0.151 | 0.149 |
| 15 | – | 613 | – | 0.078 |

(b) maze

tingency plan is then extracted from the search space by traversing the space in the forward direction using a greedy selection policy.

## Example Results

We give a sample of our experimental results for Prottle and Paragraph. For more detailed comparative results, see (Little & Thiébaux 2006). Additional results for Prottle can be found in (Little, Aberdeen, & Thiébaux 2005). Prottle and Paragraph are implemented in Common Lisp, and were both compiled using CMUCL version 19c. These experiments were performed on a machine with a 3.2 GHz Intel processor and 2 GB of RAM.

Figure shows comparative results for two problems, g-tire and maze. Their PDDL definitions are available at http://rsise.anu.edu.au/~thiebaux/benchmarks/pddl/. The planner configurations used in these experiments are: Prottle with $\epsilon = 0$ and its cost-based planning graph heuristic (PRTTL), and Paragraph with its acyclic search using either the restricted concurrency model (CA-PG) or no concurrency (NA-PG).

The objective of the g-tire problem is to move a vehicle from one location to another, where each time the vehicle moves there is a chance of it getting a flat tire. There are spare tires at some of the locations, and these can be used to replace flat tires. This problem is not concurrent. The results compare Prottle to Paragraph's acyclic search; Paragraph is faster for the earlier horizons, but Prottle scales better.

The maze problem involves a number of connected rooms and doors, some of which are locked and require a specific key to open. This problem has some potential for concurrency, although mostly of the type not allowed in composite contingency plans. None of the planner configurations fully expoit it. Paragraph scales much better than Prottle this time.

We have found that Paragraph usually out-performs Prottle. Paragraph has the best comparative performance on problems with a high forward branching factor and relatively few paths to the goal.

## Conclusion and Future Work

In Paragraph and Prottle, we have made significant progress towards our goal of producing an efficient planner that can deal with all of: concurrent durative actions, probabilistic effects, metric resources, and cost functions. We believe that both planning approaches show promise, and have a strong potential for future improvement.

The most important future improvements for Prottle include: reducing the implementation's memory usage, devising ways of efficiently extracting a greater amount of heuristic information from the planning graph, and adding support for metric resources and cost functions. Another intriguing possibility is extending Prottle's effect model (as a decision tree) to the more general graph. This might be an effective way of modelling exogenous processes.

We have many ideas for improving Paragraph's performance, in particular by adapting optimisations developed for the Graphplan framework in the deterministic setting. For example, we have observed that a small amount of control knowledge in the form of mutex invariants can make a substantial impact on efficiency. This suggests that there would also be a benefit in investigating ways of strengthening the planning graph's mutex reasoning and in incorporating explanation-based learning. But the most important future direction of this research is extending Paragraph to the probabilistic temporal setting, which will allow us to compare our two approaches in the context of probabilistic temporal planning.

## 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*.

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., and Thiébaux, S. 2006. Concurrent probabilistic planning in the graphplan framework. In *Proc. ICAPS*.

Little, I.; Aberdeen, D.; and Thiébaux, S. 2005. Prottle: A probabilistic temporal planner. In *Proc. AAAI*.

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., 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*.