Paragraph: A Graphplan-based Probabilistic Planner

Iain little
National ICT Australia & Computer Sciences Laboratory
The Australian National University
Canberra, ACT 0200, Australia

Introduction

Paragraph is a probabilistic planner that finds contingency plans that maximise the probability of reaching the goal within a given time horizon. It is capable of finding either a cyclic or acyclic solution to a given problem, depending on how it is configured. These solutions are optimal in the non-concurrent case, and optimal for a restricted model of concurrency. The concurrent case is not relevant to this discussion, and is not further discussed. A detailed description of Paragraph is given in (Little & Thiébaux 2006).

The Graphplan framework (Blum & Furst 1997) is an approach that has proven to be highly successful for solving classical planning problems. Extensions of this framework for probabilistic planning have 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. Specifically, PGraphplan finds optimal (non-concurrent) contingency plans via dynamic programming, using information propagated backwards through the planning graph to identify states from which the goal is provably unreachable. This approach takes advantage of neither the state space compression inherent in Graphplan's goal regression search, nor the pruning power of Graphplan's mutex reasoning and nogood learning. TGraphplan is a minor extension of the original Graphplan algorithm that computes a single path to the goal with a maximal probability of success; replanning could be applied when a plan's execution deviates from this path, but this strategy is not optimal.

Paragraph is an extension of the Graphplan algorithm to probabilistic planning. It enables much of the existing research into the Graphplan framework to be transfered to the probabilistic setting. Paragraph is a planner that implements some of this potential, including: a probabilistic planning graph, powerful mutex reasoning, nogood learning, and a goal regression search. The key to this framework is an efficient method of finding optimal contingency plans using goal regression. This method is fully compatible with the Graphplan framework, but is also more generally applicable.

Algorithm

To extend the Graphplan framework to the probabilistic setting, 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

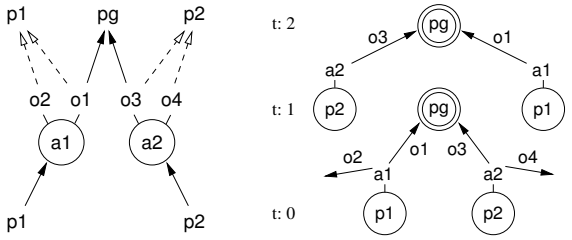

Figure 1: An action-outcome-proposition dependency graph and search space for an example problem.

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

Classical planning problems have the property that the shortest solution to a problem will not visit any given world state more than once. This is no longer true for probabilistic planning, as previously visited states can unintentionally be returned to by chance. Because of this, it is common for probabilistic planners to allow for cyclic solutions. An overview of the algorithm for producing such solutions follows. This method departs further from the `Graphplan` algorithm than the acyclic search does: fundamental to the `Graphplan` algorithm is a notion of time, which we dispense with for `Paragraph`'s cyclic search.

The cyclic search does not preserve `Graphplan`'s alternation between graph expansion and backward search: the planning graph is expanded until it levels off, and only then is the backward search performed. As there is no notion of time, the backward search is constrained only by the information represented in the final level of the levelled-off planning graph.

This cyclic search uses either a depth-first or iterative deepening algorithm. In both cases, the search uses the outcomes supporting the planning graph's final level of propositions when determining a search node's predecessors. The same principal is applied to nogood pruning: only the mutexes in the final level of the planning graph—those that are independent of time—can be safely used. An important consequence of only using universally applicable nogoods is that any new nogoods learnt from `failure` nodes are also universal. Neither search strategy is clearly superior. The depth-first search is usually preferable when searching the entire search space, as it is more likely to learn useful nogoods. A consequence of this is that there is no predictable order in which the trajectories are discovered. In contrast, the iterative deepening search will find the shortest trajectories first, which can be advantageous when only a subset of the search space might be explored.

[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`.

References

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*.

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