Probapop: Probabilistic Partial-Order Planning

Nilufer Onder Garrett C. Whelan Li Li
Department of Computer Science
Michigan Technological University
1400 Townsend Drive
Houghton, MI 49931
{nilufer,gcwhelan,lili}@mtu.edu

Abstract

We describe Probapop, a partial-order probabilistic planning system. Probapop is a blind (conformant) planner that finds plans for domains involving probabilistic actions but no observability. The Probapop implementation is based on Vhpop, a partial-order deterministic planner written in C++. The Probapop algorithm uses plan graph based heuristics for selecting a plan from the search queue, and probabilistic assessment heuristics for selecting a condition whose probability can be increased.

Introduction

Probapop[1] is a *conformant probabilistic planner* (term used in (Hyafil & Bacchus 2003)). In this paradigm, the actions and the initial state can be *probabilistic*, i.e., they can have several possible outcomes annotated by a probability of occurrence. In addition, the planning problem is *conformant* i.e., the agent cannot observe the environment. The objective is to find a minimal sequence of steps that will take an agent from an initial set of states to a specified goal state within a specified threshold probability. Note that while the assumption of blind agents is not true in general, it is useful to incorporate conformant planning methods because sensing might be expensive, not reliable, or not available. We leave contingency planning, e.g., (Majercik & Littman 1999; Onder & Pollack 1999; Hansen & Feng 2000; Karlsson 2001) and other paradigms that assume non-probabilistic effects, e.g.,(Ferraris & Giunchiglia 2000; Bertoli, Cimatti, & Roveri 2001) outside the current implementation of Probapop.

Our work is motivated by the incentive to have partial-order planning as a viable option for conformant probabilistic planning. The primary reason is that partial-order planners have worked very well with lifted actions which are useful in coding large domains in a compact way. Second, due to its least commitment strategy in step ordering, partial-order planning (POP) produces plans that are highly parallelizable. Third, planners that can handle rich temporal constraints have been based on POP algorithms (Smith, Frank, & Jonsson 2000).

Our basic approach is to form base plans by using deterministic partial-order planning techniques, and then to estimate the best way to improve these plans. Recently Repop (Nguyen & Kambhampati 2001) and Vhpop (Younes & Simmons 2002) planners have demonstrated that the very heuristics that speed up non-partial-order planners can be used to scale up partial-order planning. We show that these distance based heuristics (McDermott 1999; Bonet & Geffner 1999) as implemented using "relaxed" plan graphs can be employed in probabilistic domains. These, coupled with selective plan improvement heuristics result in significant improvement. As a result, *Probapop* enjoys the soundness, completeness, and least-commitment properties of partial-order planning and makes partial-order planning feasible in probabilistic domains.

Probapop and Partial-Order Planning

For partial-order probabilistic planning, we implemented the Buridan (Kushmerick, Hanks, & Weld 1995) probabilistic planning algorithm on top of Vhpop (Younes & Simmons 2002), a recent partial-order planner. A partially ordered plan π is a is 6-tuple, <`STEPS, ORD, BIND, LINKS, OPEN, UNSAFE`>, representing sets of ground actions, ordering constraints, binding constraints, causal links, open conditions, and unsafe links, respectively. An ordering constraint $S_i \prec S_j$ represents the fact that step S_i precedes S_j. A *causal link* is a triple $< S_i, p, S_j >$, where S_i is the *producer*, S_j is the consumer and p represents the condition supported. An *open condition* is a pair $< p, S >$, where, p is a condition needed by step S. A causal link $< S_i, p, S_j >$ is *unsafe* if the plan contains a *threatening* step S_k such that S_k has $\overline{p}$ among its effects, and

[1]This work has been supported by a Research Excellence Fund grant from Michigan Technological University.

S_k may intervene between S_i and S_j. Open conditions and unsafe links are collectively referred to as *flaws*. A *planning problem* is a triple (I, G, t), where, the initial state I is a probability distribution over states, G is a set of literals that must be true at the end of execution, and t is a probability threshold. The planner must find a plan that takes the agent from I to G with a probability $\geq t$. If several plans have the same probability of success, then the one with the least number of steps is preferred.

The Probapop algorithm shown in Fig. 1 first constructs an initial plan by forming I and G into initial and goal steps, and then refines the plans in the search queue until it finds a solution plan that meets or exceeds the probability threshold. Plan refinement operations involve repairing flaws. An open condition can be closed by adding a new step from the domain theory, or reusing a step already in the plan. An unsafe link is handled by the *promotion*, *demotion*, or *separation* (lifted actions are used) operations, or by *confrontation*(Penberthy & Weld 1992) which involves commitment to non-threatening effects.

```
function PROBAPOP (initial, goal, t)
returns a solution plan, or failure
   plans ← MAKE-MINIMAL-PLAN(initial, goal)
   loop do
      if plans is empty then return failure
      plan ← REMOVE-FRONT(plans)
      if SOLUTION?(plan, t) then return plan
      plans ← MERGE(plans, REFINE-PLAN(plan))
   end

function REFINE-PLAN (plan)
returns a set of plans (possibly null)
   if FLAWS(plan) is empty then
      plan ← REOPEN-CONDITIONS(plan)
   flaw ← SELECT-FLAW(plan)
   if flaw is an open condition then choose:
         return REUSE-STEP(plan, flaw)
         return ADD-NEW-STEP(plan, flaw)
   if flaw is a threat then choose:
         return DEMOTION(plan, flaw)
         return PROMOTION(plan, flaw)
         return SEPARATION(plan, flaw)
         return CONFRONTATION(plan, flaw)
```

Figure 1: The probabilistic POP algorithm.

The search is conducted using an A* algorithm guided by a *ranking function* f. As usual for a plan π, $f(\pi) = g(\pi) + h(\pi)$, where $g(\pi)$ is the cost of the plan, and $h(\pi)$ is the estimated cost of completing it. In Probapop, g reflects the number of steps in a plan, h represents the estimated number of steps to complete a plan. Both are weighted by the probability of success of the overall plan. The ranking function is used at the MERGE step to order the plans in the search queue such that the plan that ranks best is at the beginning of the queue. We term a plan for which `OPEN` = `UNSAFE` = $\emptyset$ as a *quasi-complete* plan. A quasi-complete plan is not a solution if it does not meet the probability threshold. Probapop can be viewed as first choosing a plan to improve using the ranking function, then choosing a way to improve the plan, and finally choosing a way to implement the improvement. These phases do not have to follow strictly or work on the same plan. After the successors of a plan are generated, the ranking function might gear the search toward other plans in the search queue. In the next section, we describe the heuristics used.

Distance Based Ranking and Selective Reopening in Probapop

The Vhpop deterministic partial order-planner described in (Younes & Simmons 2002) implements the *ADD heuristic* to provide an estimate of the total number of new actions needed to close all the open conditions. Before starting to search, the planner builds a planning graph (Blum & Furst 1997) which has the literals in the initial state in its first level, and continues to expand it until it reaches a level where all the goal literals are present. Vhpop's *ADD* heuristic achieves good performance by computing the step cost of the open conditions from the planning graph, i.e., $h(\pi) = h_{add}(OPEN(\pi))$. The cost of achieving a literal q is the level of the first action that achieves q: $h_{add}(q) = min_{a \in GA(q)} h_{add}(a)$ if $GA(q) \neq \emptyset$, where $GA(q)$ is an action that has an effect q. Note that $h_{add}(q)$ is 0 if q holds initially, and is ∞ if q never holds. The *level* of an action is the first level its preconditions become true: $h_{add}(a) = 1 + h_{add}(PREC(a))$.

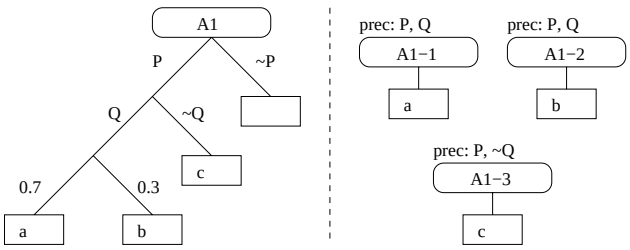

Figure 2: Probabilistic action A1 is split into deterministic actions A1-1, A1-2, and A1-3.

In order to be able to use *ADD* with probabilistic effects, one would need to split into as many plan graphs as there are leaves in a probabilistic action. To avoid this, we split each action in the domain theory into as many deterministic actions as the number of nonempty effect lists each representing a possible way the original

action would work (Fig. 2). By using the split actions, we can compute a good estimate of the number of actions needed to complete a plan. While the plan graph uses split actions, the plans in the search queue always contain the full original action so that the planner can correctly assess the probability of success. Our current ranking function uses this assessment to prefer plans with higher probability of success, and if there is a tie, the plan with less number of steps is preferred.

An important distinction between deterministic partial-order planning and probabilistic partial-order planning is multiple support for plan literals. In the deterministic case, an open condition is permanently removed from the list of flaws once it is resolved. In the probabilistic case, it can be *reopened* so that the planner can search for additional steps that increase the probability of the literal. We address this problem by employing *selective reopening* (SR) where we select a random total ordering of the plan; look at the state distribution after the execution of each step; and reopen only those conditions that are not guaranteed to be achieved. While plan assessment is costly for probabilistic plans, this is a one time cost incurred only on quasi-complete plans and we have observed that the benefit of avoiding extra plans in the search space far exceeds the computational overhead incurred.

It is important to note that neither the split actions nor the selective reopening technique change the base soundness and completeness properties of the Buridan algorithm. The split actions are only used in the relaxed plan graph, and the reopening technique does not block any alternatives from being sought as they would already be covered by a plan in the search queue.

Conclusion and Future Work

We presented Probapop, a partial-order probabilistic planner. We described distance-based and probabilistic condition based heuristics for partial-order probabilistic planning. We informally noted that neither the split actions nor the selective reopening technique change the base soundness and completeness properties of the Buridan algorithm.

Probapop is different than policy generating planners such as Spudd(Hoey *et al.* 1999) and Gpt(Bonet & Geffner 2000) in the sense that it generates plans. Given a planning problem, Probapop returns a sequence of steps that achieve the goal with a probability that meets or exceeds the specified threshold. The plan generated does not rely on sensing actions in order to be executed. Our future work involves adding the capability to deal with partially observable domains to Probapop.

References

Bertoli, P.; Cimatti, A.; and Roveri, M. 2001. Heuristic search + symbolic model checking = efficient conformant planning. In *Proc. 18th Intl. Joint Conf. on Artificial Intelligence*, 467–472.

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

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *Proc. 5th European Conf. on Planning (ECP'99)*.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proc. 5th Intl. Conf. AI Planning & Scheduling*, 52–61.

Ferraris, P., and Giunchiglia, E. 2000. Planning as satisfiability in nondeterministic domains. In *Proc. 17th Nat. Conf. Artificial Intelligence*, 748–754.

Hansen, E. A., and Feng, Z. 2000. Dynamic programming for POMDPs using a factored state representation. In *Proc. 5th Intl. Conf. AI Planning & Scheduling*, 130–139.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proc. 15th Conf. Uncertainty in AI*.

Hyafil, N., and Bacchus, F. 2003. Conformant probabilistic planning via csps. In *Proc. 13th Intl. Conf. Automated Planning & Scheduling*.

Karlsson, L. 2001. Conditional progressive planning under uncertainty. In *Proc. 18th Intl. Joint Conf. on Artificial Intelligence*, 431–436.

Kushmerick, N.; Hanks, S.; and Weld, D. S. 1995. An algorithm for probabilistic planning. *AIJ* 76:239–286.

Majercik, S. M., and Littman, M. L. 1999. Contingent planning under uncertainty via stochastic satisfiability. In *Proc. 16th Nat. Conf. Artificial Intelligence*, 549–556.

McDermott, D. 1999. Using regression-match graphs to control search in planning. *AIJ* 109(1-2):111–159.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *Proc. 18th Intl. Joint Conf. on Artificial Intelligence*, 459–464.

Onder, N., and Pollack, M. E. 1999. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In *Proc. 16th Nat. Conf. Artificial Intelligence*, 577–584.

Penberthy, J. S., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. Third Intl. Conf. Principles of Knowledge Representation & Reasoning*, 103–114.

Smith, D. E.; Frank, J.; and Jonsson, A. K. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15(1).

Younes, H. L., and Simmons, R. G. 2002. On the role of ground actions in refinement planning. In *Proc. 6th Intl. Conf. AI Planning & Scheduling*, 54–61.