Revisiting Partial-Order Probabilistic Planning

Nilufer Onder
Department of Computer Science
Michigan Technological University
1400 Townsend Drive
Houghton, MI 49931
nilufer@mtu.edu

Li Li
Department of Computer Science
Michigan Technological University
1400 Townsend Drive
Houghton, MI 49931
lili@mtu.edu

Abstract

We present a partial-order probabilistic planning algorithm that adapts plan-graph based heuristics implemented in Repop. We describe our implemented planner, Reburidan, named after its predecessors Repop and Buridan. Reburidan uses plan-graph based heuristics to first generate a base plan. It then improves this plan using plan refinement heuristics based on the success probability of subgoals. Our initial experiments show that these heuristics are effective in improving Buridan significantly.

Introduction

During the last years, deterministic planning algorithms have demonstrated significant progress in dealing with large problems. Most notable scaling up has been observed with plan-graph based, constraint satisfaction problem (CSP) based, or state space based planning paradigms rather than partial-order planners which were previously dominant in planning research for several decades. In particular, the number of steps that can be synthesized into a plan has increased to the order of a hundred steps making it conceivable that realistic problems can be solved using AI planning techniques. On the other hand, most realistic problems require an agent to operate in an uncertain environment, and it is unfortunate that a planner that can deal with large, complex, non-deterministic domains has not emerged despite ubiquitous need (Smith, Frank, & Jonsson 2000; Wilkins & desJardins 2001).

Recently, it was demonstrated that the very heuristics that speed up non-partial-order planners can be used to scale up partial-order planning (Nguyen & Kambhampati 2001). It is argued that in deterministic domains, the partial-order planning paradigm might be desirable due to two main reasons. First, due to its least commitment strategy, partial-order planning (POP) produces plans which offer more execution flexibility as compared to other planning paradigms. In particular, steps have precedence relations between them only if there is a causal connection between them, or they must be ordered to make the plans correct. The execution flexibility offered by partial order planners is significant in multi-agent domains because the plans are highly parallelizable making the use of several agents possible. Second, the POP framework has been used for domains which require reasoning about time. In particular, planners that can handle rich temporal constraints have been based on POP algorithms.

Furthermore, when a planning domain has uncertainty, optimization concerns come into the picture either explicitly or implicitly. When the optimization requirement is explicit, i.e., the planner needs to find a plan that makes the best use of the resources available, all the planning paradigms are faced with exponential explosion in the search space. On the other hand, there are domains where multiple criteria must be considered, but an optimization problem cannot be explicitly specified due to the lack of an objective function. Obviously, in probabilistic domains, optimality can no more be measured by the number of steps, because a long plan might have a larger probability of success than a shorter one. Because all base plans are candidates for being improved to become a solution plan, an iterative approach might be desirable, so that an objective function can also be iteratively formulated as options become clearer. In implementing such an approach, constructing a least commitment plan as the base plan is advantageous because it results in the most compact representation for further iterations.

We therefore believe there is great incentive to explore the ways for improving the speed of partial order probabilistic plans. In this paper, we explore these approaches by demonstrating that plan graph analysis and other heuristics implemented in the Repop system (Nguyen & Kambhampati 2001) can be applied to probabilistic partial-order planning to form a partially ordered base plan. These, coupled with selective plan improvement heuristics result in significant improvement over Buridan, a partial-order planner (Kushmerick, Hanks, & Weld 1995). In addition, by using a partial-order plan representation, we can avoid splitting the plans into several "branches" as uncertainty is introduced. The result is a planning algorithm that enjoys the soundness, completeness, and flexible execution properties of probabilistic partial-order planning, and benefits from speed-up heuristics of other planning paradigms.

The purpose of this paper is to show preliminary experiments with our Reburidan probabilistic planning system, named after its predecessors Repop and Buridan. In the remainder of this paper, we first provide background on probabilistic planning. We then describe our planning algorithm, Reburidan, which has been named after its predecessors Repop and Buridan. We describe the heuristics used and provide empirical results demonstrating their effectiveness. We conclude with a summary and directions for future work.

Background

We begin by providing a brief description of partial-order probabilistic planning, and we build on the representation first developed for Buridan (Kushmerick, Hanks, & Weld 1995, p. 247).

A partially ordered plan P is a is 5-tuple, <`STEPS, ORD, LINKS, OPEN, UNSAFE`>, where `STEPS` is a set of ground actions, `ORD` is a set of ordering constraints, `LINKS` is a set of causal links, `OPEN` is a set of open conditions, and `UNSAFE` is a set of unsafe links.

The steps come from a domain theory which contains *actions* similar to the STRIPS representation with their precondition and effect lists except for the fact that preconditions are called *triggers*, and the effects are probabilistic. An *action* is a set of triples $\{< t_1, \rho_1, e_1 >, \ldots, < t_n, \rho_n, e_n >\}$, where each t_i is a set of literals called a *trigger*, e_i is a set of literals called the *effects*, and ρ_i is the probability that the effects in e_i will take place if all the literals in t_i hold at the time of execution. By convention, the triggers are exhaustive and mutually exclusive, and the probabilities for each distinct trigger add up to 1. Figure 1 depicts an example probabilistic action taken from the logistics domain. The labels on the arcs are the triggers, and the probabilities, and the rectangular boxes contain the effects. The empty boxes denote that the action has no effect under the circumstances listed.

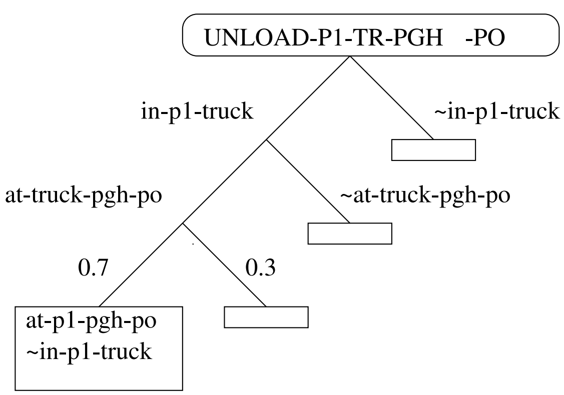

Figure 1: The UNLOAD action succeeds only with probability 0.7 when the trigger conditions are true.

An ordering constraint is of the form $S_i \prec S_j$ and represents the fact that step S_i precedes step S_j. A *causal link* is a triple $< S_i, p, S_j >$, where S_i is the *producer* step, S_j is the consumer step and p is a literal that represents the condition supported. (We refer the reader to (Kushmerick, Hanks, & Weld 1995) for further details regarding probabilistic representation.) An *open condition* is a pair $< p, S >$, where, p is a literal representing 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 S_k may intervene between S_i and S_j. Note that a probabilistic action may have several sets of effects and an action will be considered to be a threatener as long as one set of effects contains $\overline{p}$. Open conditions and unsafe links in a plan are collectively referred to as *flaws*.

A planning problem is a triple (I, G, t), where, I is a probability distribution over states representing the initial state, G is a set of literals that must be true at the end of plan execution, and t is a probability threshold.

The Buridan algorithm operates as follows: it constructs an initial plan by forming I and G into initial and goal steps respectively, and then refines the plans in the search queue until it finds a solution plan that meets or exceeds the probability threshold. We term a plan for which `OPEN` $= \emptyset$ and `UNSAFE` $= \emptyset$ as a *quasi-complete* plan. In probabilistic planning, a quasi-complete plan might fail to be a solution if it does not meet the probability threshold.

Plan refinement operations involve repairing flaws: closing an open condition, or handling an unsafe link. 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* and *demotion* operations, or by *confrontation*. Confrontation was introduced in UCPOP(Penberthy & Weld 1992), and involves marking commitment to those conditional or probabilistic effects of an action that do not include the threatening condition.

The search is conducted using an $A*$ algorithm guided by a *ranking function* which provides the f value. In the next section, we describe how Repop's relax heuristic can be adapted to provide a better ranking function for Buridan.

Repop's Relax Heuristic

Nguyen and Kambhampati describe a combination of heuristics that bring the performance of Ucpop to the level of the state-of-the-art planners(Nguyen & Kambhampati 2001). One such heuristic—the *relax heuristic*—provides an estimate of the total number of new actions needed to close all the open conditions. The relax heuristic involves ignoring the negative interactions among the steps in the plan and building a planning graph akin to Graphplan's planning graph (Blum & Furst 1997) to compute distance-based heuristics (Ghallab & Laurelle 1994; McDermott 1999; Bonet & Geffner 2001).

Before starting to search, Repop first builds a planning graph 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 without mutex relationships between them, and the plan graph is static.

Suppose that a partially ordered plan P_i in the search space has a set of literals $L = \{l_1, \ldots, l_i, \ldots, l_n\}$ as the conditions in `OPEN`. To compute the estimated cost of achieving all the conditions in L, Repop uses the plan graph to find the last literal in L, say l_i, to be achieved by the plan. Suppose that S_j is the step that achieves l_i. Then one can write a recurrence relation for the cost of achieving all the conditions in L:

`cost(`L`) = new-step(`S_j`) +`
`cost (`$L \cup$ `preconds(`S_j`) - effects (`S_j`))`

This formula is recursively based on the preconditions of S_j and reaches its base condition when L is the set of conditions in the initial state yielding a cost of 0. `New-step` evaluates to 1 if S_j is not among the steps in the plan, and to 0 otherwise.

The `cost` value is used to compute the rank of a plan P as follows:
rank(P) = | `STEPS(` P`)`| + w * `cost(conds ( OPEN))`, where, w is an adjustable parameter used to increase the greediness of the search.

Relax Heuristic in Buridan

As it can be seen, the computation of the estimated cost depends on building a plan graph. In order to account for probabilistic effects, one would need to split the plan graph into as many plan graphs as there are leaves in a probabilistic action. To avoid this, we rest on the observation that the search space of a probabilistic partial-order planner contains two kinds of plans. The first kind is a quasi-complete plan which does not have any open conditions or unsafe links. If a quasi-complete plan meets the probability threshold, then it is a solution plan. If it does not meet the threshold, it might be possible to improve it. The second kind is an incomplete plan which has flaws to be taken care of. Therefore, one can view plan refinement as a two phase process. The first phase consists of making the plan quasi-complete, and the second phase consists of further improving the quasi-complete plan so that it meets the probability threshold.

While building a quasi-complete plan, we can temporarily ignore the actual probability numbers, and concentrate on repairing the flaws. When the probability numbers are ignored, the planning problem becomes deterministic, and Repop's plan-graph based heuristics can be used.

To facilitate this, we split each action in the domain theory into as many deterministic actions as the number of nonempty effect lists. Each new action represents a possible way the original action would work (Fig. 2). For incomplete plans, we use the deterministic actions and plan-graph based heuristics. Once the plan becomes quasi-complete, we revert the operators back to their probabilistic originals and let the planner work with heuristics that are not based on the plan graph.

This approach has three advantages:

1. The search queue is uniform in the sense that both quasi-complete plans and incomplete plans are partial-order plans. As opposed to using a non partial-order planner to come up with a quasi-complete plan, this offers better potential for integrating an expressive language for use with more complicated domains such as the ones described in (Smith, Frank, & Jonsson 2000).
2. The quasi-complete plans can be returned at any time as intermediate solutions (*anytime* algorithms).
3. Both the quasi-complete plans and the solution plans are highly parallelizable due to partially ordered representation.

Reopening Conditions

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.

When Buridan retrieves a quasi-complete plan from the search queue, it indiscriminantly reopens all the previously closed conditions resulting in needless expansion of the search space. We address this problem by employing *selective reopening* (SR). In particular, we select a random total ordering of the plan and look at the state distribution after the execution of each step to reopen only the conditions involving literals that are not guaranteed to be achieved.

As an example, consider the situation depicted in Fig. 3, where, <T, STEPi>, and <a, STEPj> are open conditions that are now closed by causal links. Let PROB(a,S) denote the probability of condition a just before the execution of step S. We reopen only those previously closed conditions c, where,

- c is an effect needed for STEPi, and PROB(c,STEPi) <1, or
- c is a trigger of a STEPi, and PROB(c,STEPi) <1 and c is a trigger for an effect a needed for STEPj, and PROB(a,STEPj) <1.

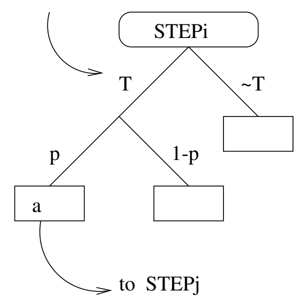

Figure 3: <T,STEPi> and <a,STEPj> are open conditions that are now closed. The arcs represent causal links.

The rationale behind the first case is that there is no need for additional support for an effect that will be achieved with probability 1. The rationale for the second case is that there is no need for additional support for a trigger if it already is expected to be achieved with probability 1, or the effect it enables will be achieved with probability 1.

As we show in the next section, the benefit of avoiding extra plans in the search space far exceeds the computational overhead incurred.

Empirical Results

We have conducted a set of preliminary experiments by using the transportation and robot domains which were shown to benefit from Repop's heuristics both in terms of speed up and plan quality. We ran all the experiments using Allegro Lisp on a 550 MHz Linux machine.

We coded probabilistic actions for an increasing number of packages (logistics-1 has 1 package, and logistics-5 has 5

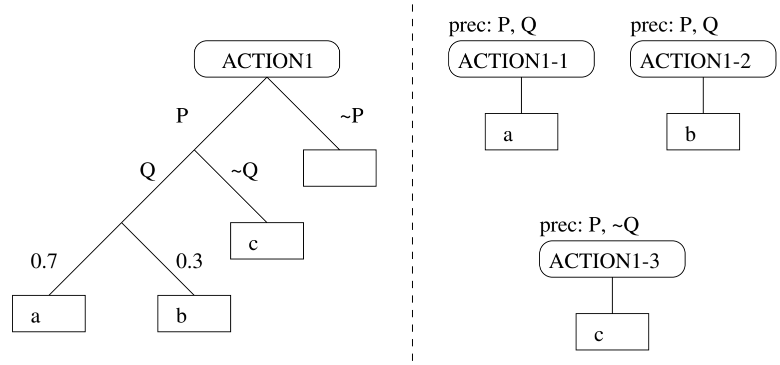

Figure 2: Probabilistic action ACTION1 is split into deterministic actions ACTION1-1, ACTION1-2, and ACTION1-3.

packages), and objects to be delivered (gripper-2 has 2 objects, and gripper-4 has 4). In Table 1, we tabulate the improvement in the performance of Buridan as our heuristics are added. In these experiments, the probability threshold is set to 0.6. Repop refers to the Repop heuristics and SR refers to selective reopening. For the cases marked with a "*", no solution was found within 20 minutes of processor time, or within 10,000 plans in the search space. For logistics-5, Buridan + Repop was only able to find a quasi-complete plan, therefore the run time is shown in parentheses. The largest number of steps was 45. Buridan can solve the simple logistics problem which yields a plan with 4 steps, but fails to solve the larger problems within the time allotted. The SR heuristic is more effective in the logistics domain due to the large number of operators instantiated.

In Table 2, we show the run times for the best performing combination, i.e., Buridan+Repop+SR, as the probability threshold is increased. While selective reasoning can be observed to be helpful, the steep increase in run times shows that more powerful heuristics for plan improvement are needed. We are currently looking into techniques for incorporating probability of success into the ranking function as well as expanding the plan graph beyond the point where all the goal conditions are satisfied.

Related Work

Early work has been done in improving Buridan with probability based heuristics(Blythe 1995). The recent trend is to augment the state-of-the-art planners with reasoning under uncertainty. Work has been done in stochastic satisfiability based planning in probabilistic domains (Majercik & Littman 1998), and in non-probabilistic domains (Ferraris & Giunchiglia 2000); probabilistic planning in the Graphplan framework (Blum & Langford 1999); non-probabilistic planning under uncertainty using state space search (Bertoli, Cimatti, & Roveri 2001); and non-observable Markov Decision Process (MDP) based planners (Boutilier, Dean, & Hanks 1999).

Non-probabilistic conformant planning is also a "blind" planning technique which has been explored in (Goldman & Boddy 1996; Smith & Weld 1998; Bertoli, Cimatti, & Roveri 2001; Ferraris & Giunchiglia 2000). Conformant planners generate plans which cover all possible situations by finding alternative actions. Probabilistic planners can in addition use the same action repeatedly to increase the probability of success.

There has also been work in integrating conditional and probabilistic planning in STRIPS domains (Draper, Hanks, & Weld 1994; Majercik & Littman 1999; Onder & Pollack 1999; Hansen & Feng 2000; Karlsson 2001), and in more expressive domains allowing derived and functional effects (Ngo, Haddawy, & Nguyen 1998). Most likely, no planning paradigm will be superior in all kinds of domains. Consequently, there is no consensus on a set of benchmark set of problems to allow comparison of planners. On the positive side, new application domains are emerging as the planning technology advances. We will be providing a seed set of benchmark problems with the release of Reburidan.

Conclusion

In this paper, we described powerful heuristics for probabilistic partial-order planning. In designing our heuristics, we relied upon plan graph analysis techniques implemented in Repop, and probabilistic analysis to separate the conditions that are worthwhile for added support. Our experiments show substantial speedup over Buridan, and we believe that the basic concepts can be carried over to more complicated domains.

We are currently working on expanding our experiments to other probabilistic domains and performing comparisons to other probabilistic planners such as Maxplan (Majercik & Littman 1998), PGraphplan (Blum & Langford 1999), SPUDD (Hoey *et al.* 1999) and GPT (Bonet & Geffner).

Acknowledgments

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

References

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

Problem	Buridan	Buridan + SR	Buridan + Repop	Buridan + Repop + SR
logistics-simple	1,280	860	1,241	1,231
logistics-1	*	*	12,840	11,040
logistics-2	*	*	69,520	47,340
logistics-3	*	*	237,123	125,810
logistics-5	*	*	(305,580)	202,620
gripper-2	*	*	6,090	5,090
gripper-3	*	*	20,390	18,030
gripper-4	*	*	107,430	106,150

Table 1: Run times in msecs.

Problem	t=0.5	t=0.6	t=0.8	t=0.9
logistics-simple	950	1231	9.950	156,085
logistics-1	10,130	11,040	51,452	1,000,810
logistics-2	44,850	47,340	165,420	2,937,600
logistics-3	120,800	125,810	471,500	> 1 hr.
logistics-5	325,810	202,620	1,414,410	> 1 hr.
gripper-2	4,230	5,090	15,510	246,010
gripper-3	10,430	18,030	123,450	> 1 hr.

Table 2: Run times in msecs as the probability threshold (t) is increased.

Blum, A. L., and Furst, M. L. 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 *Proceedings of the 5th European Conference on Planning (ECP'99)*.

Blythe, J. 1995. The footprint principle for heuristics for probabilistic planners. In *Proceedings of the European Workshop on Planning*.

Bonet, B., and Geffner, H. GPT: A tool for planning with uncertainty and partial information. In *IJCAI-2001 Workshop on Planning with Incomplete Information*.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems*, 31–36.

Ferraris, P., and Giunchiglia, E. 2000. Planning as satisfiability in nondeterministic domains. In *Proceedings of the 17th National Conference on Artificial Intelligence*, 748–754.

Ghallab, M., and Laurelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems*, 61–67.

Goldman, R. P., and Boddy, M. S. 1996. Expressive planning and explicit knowledge. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems*, 110–117.

Hansen, E. A., and Feng, Z. 2000. Dynamic programming for POMDPs using a factored state representation. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*, 130–139.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*.

Karlsson, L. 2001. Conditional progressive planning under uncertainty. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 431–436.

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

Majercik, S. M., and Littman, M. L. 1998. Using caching to solve larger probabilistic planning problems. In *Proceedings of the 15th National Conference on Artificial Intelligence*.

Majercik, S. M., and Littman, M. L. 1999. Contingent planning under uncertainty via stochastic satisfiability. In *Proceedings of the 16th National Conference on Artificial Intelligence*, 549–556.

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

Ngo, L.; Haddawy, P.; and Nguyen, H. 1998. A modular structured approach to conditional decision-theoretic planning. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems*, 111–118.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 459–464.

Onder, N., and Pollack, M. E. 1999. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In *Proceedings of the 16th National Conference on Artificial Intelligence*, 577–584.

Penberthy, J. S., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 103–114.

Smith, D. E., and Weld, D. S. 1998. Conformant Graphplan. In *Proceedings of the 15th National Conference on Artificial Intelligence*, 889–896.

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

Wilkins, D. E., and desJardins, M. E. 2001. A call for knowledge-based planning. *AI Magazine* 22(1):99–115.