

# Cost Sensitive Conditional Planning

**Daniel Bryce & Subbarao Kambhampati**

Department of Computer Science and Engineering  
Arizona State University, Brickyard Suite 501  
699 South Mill Avenue, Tempe, AZ 85281  
{dan.bryce, rao}@asu.edu

## Abstract

While POMDPs provide a general platform for conditional planning under a wide range of quality metrics they have limited scalability. On the other hand, uniform probability conditional planners scale very well, but many lack the ability to optimize plan quality metrics. We present an innovation to planning graph based heuristics that helps uniform probability conditional planners both scale and generate high quality plans when using actions with non uniform costs. We make empirical comparisons with two state of the art planners to show the benefit of our techniques.

## Introduction

When agents have uncertainty about their state, they need to formulate conditional plans, which attempt to resolve state-uncertainty with sensing actions. This problem has received attention in both the uncertainty in AI (UAI) and automated planning communities. From the UAI perspective, finding such conditional plans is a special case of finding policies for FOMDPs in the fully observable case, and POMDPs in the partially observable case. The latter is of more practical use, although much harder computationally [Madani *et al.*, 1999; Littman *et al.*, 1998]. The emphasis in the UAI community has been on finding optimal policies under fairly general conditions. However the scalability of the approaches has been very limited. In the planning community, conditional planning has been modelled as search in the space of belief states (which can be seen as a way of characterizing state uncertainty in terms of uniform probability over a set of states). Several planners have been developed—eg. MBP [Bertoli *et al.*, 2001], and PKSPlan [Petrick and Bacchus, 2002]—which model conditional plan construction as an and/or search. These approaches are more scalable than the MDP-based approaches<sup>1</sup>, but are often insensitive to the cost/quality information. Indeed, in the presence of actions with differing costs, planners such as MBP (aside from using in-admissible heuristics) can generate plans of arbitrarily low quality, attempting to insert sensing actions without taking their cost into consideration.

In this paper, we describe a way of extending state of the art conditional planners to make them more sensitive to

cost/quality information. Our idea is to adapt the type of cost-sensitive reachability heuristics that have proven to be useful in classical and temporal planning [Do and Kambhampati, 2003]. Straightforward adaptation unfortunately proves to be infeasible. This is because in the presence of state uncertainty, we will be forced to generate multiple planning graphs (one for each possible state) and reason about reachability across all those graphs [Bryce and Kambhampati, 2004]. This can get prohibitively expensive—especially for forward searching planners which need to do this analysis once at each search node.

The main contribution of this paper is a way to solve this dilemma. In particular, we propose a novel way of generating reachability information with respect to belief states without computing multiple graphs. Our approach, called the labelled uncertainty graph (*LUG*) [Bryce *et al.*, 2004], symbolically represents multiple planning graphs, one for each state in our belief, within a single planning graph. Loosely speaking, this single graph unions the support information in explicit multiple graphs and pushes the disjunction, describing sets of possible worlds (states in a belief), into “labels” ( $\ell$ ). The graph is built using labels, for sets of worlds, to annotate the planning graph vertices. A label on a vertex signifies the states of our belief that reach the vertex.

To take cost information into account, we describe a method for propagating cost information over the *LUG*, giving us a cost-propagated *LUG* (*CLUG*). The (previously mentioned) labels tell us when graph vertices are reachable, but they do not indicate the associated reachability cost. We could track a single cost for the entire set of worlds represented by a label, but this would lose information about differing costs for subsets of the worlds. Tracking a cost for each subset of worlds is also problematic because there are an exponential number of subsets. Instead we track cost over fixed partitions of world sets. This *CLUG* is used as the basis for doing reachability analysis. In particular, we extract relaxed plans from it (as described in [Bryce *et al.*, 2004]), and use the cost information to help select low cost relaxed plans. Our results show that cost-sensitive heuristics improve plan quality.

We proceed by describing our representation and our planner *POND*, then introduce our planning graph generalization called the *CLUG*, and relaxed plan extraction procedure. We then do an empirical study of the techniques

---

<sup>1</sup>Their scalability is partly because the complexity is *only* 2-EXP-complete [Rintanen, 2004]!

within our planner and compare with two state of the art conditional planners MBP [Bertoli *et al.*, 2001] and GPT [Bonet and Geffner, 2000]. We end with a conclusion and directions for future work, with emphasis on non-uniform uncertainty.

## Representation & Search

*POND* uses progression search to find strong plans, under the assumption of partial observability. A strong plan guarantees that after a finite number actions executed from any of the many possible initial states, all resulting states will satisfy the goals. The plans are directed acyclic graphs. We define a plan’s quality as the expected execution cost (i.e. average path cost), under uniform probability.

*POND* searches in the space of uniform probability belief states, a technique first described by Bonet and Geffner [2000]. The planning problem  $P$  is a tuple  $\langle D, BS_I, BS_G \rangle$ , where  $D$  is a domain,  $BS_I$  is the initial belief state, and  $BS_G$  is the goal belief state. Belief states are essentially propositional formulas whose models are states. We represent belief states with BDDs. The domain  $D$  is a tuple  $\langle F, A \rangle$ , where  $F$  is a set of all fluents and  $A$  is a set of actions. Actions have cost and are either causative or observational, with a set of conditional effects or observation formulae as outcomes, respectively.

We use top down AO\* search [Nilsson, 1980], in the *POND* planner to generate conditional plans. In the search graph, the nodes are belief states and the hyper-edges are actions. We need AO\* because the application of a sensing action to a belief state in essence partitions the belief state. We use hyper-edges for actions because sensory actions have several outcomes, all if any of which must be included in a solution. The cost model for our plans is the expected execution cost so we use an expectation over the children of a hyper-edge to choose a node’s best action.

### Cost Labelled Uncertainty Graph (*CLUG*)

To guide search, we use a relaxation of conditional planning to estimate the conditional plan’s suffix for each search node. The relaxation measures the cost needed to support the goal by ignoring mutexes between actions, and ignoring sensory actions. While our relaxation does not include sensory actions, the search reasons about the cost of sensing at the current search node. Our heuristic reasons about the transition cost between two sets of states, giving a belief state cost measure [Bryce and Kambhampati, 2004]. The belief state cost measure is a generalization of state cost measures used in classical planning that estimate path costs in the search graph. Following [Bryce and Kambhampati, 2004], we compute the belief state cost measure to estimate the cost of co-transitioning all states in our current belief state to a state in the goal belief state. While we estimate the same metric, we compute it within a single planning graph called the *LUG* [Bryce *et al.*, 2004] opposed to using a planning graph for every state in our belief state. The *LUG* was originally developed for unit cost actions, and here we define a generalization of cost propagation techniques for the *LUG*.

### *CLUG* Construction

We present the *CLUG*, a single planning graph that uses annotations on vertices (actions and literals) to reflect various assumptions about how a vertex ( $v$ ) is reached. Specifically we use two annotations, a label ( $\ell_k(v)$ ), which denotes the models of our current (source) belief  $BS_s$  that reach the vertex at level  $k$  and a cost vector reflecting an estimate of the cost of reaching the vertex from different models of the source belief. The annotations help us implicitly represent the vertices common to several of the multiple planning graphs in a single planning graph. The labels for the initial layer literals are used to label the actions and effects they support, which in turn label the literals they support. The use of labels is based on the intuition that (i) actions and effects are applicable in the possible worlds for which their conditions are reachable and (ii) a literal is reachable in all possible worlds where it is given as an effect.

***CLUG***: A levelled graph, where a level  $k$  contains three layers, the literal  $\mathcal{L}_k$ , action  $A_k$ , and effect  $\mathcal{E}_k$  layers. The construction of the *LUG* is with respect to the actions in  $A$  and a source belief state  $BS_s$ . Each *LUG* vertex is a triple  $\langle v, \ell_k(v), c_k(v) \rangle$ , where the  $v$  is an action  $a$ , effect  $\varphi_j$ , or literal  $l$ ,  $\ell_k(v)$  is its label, and  $c_k(v)$  is a cost vector.

**Labels**: A label  $\ell_k(v)$  of a vertex  $v$  is a propositional formula where each model of it is a state  $S_s \in \mathcal{M}(BS_s)$ . For any such  $S_s$ , a classical planning graph built from  $S_s$  contains  $v$  in level  $k$ .

Without considering cost, as in the *LUG*, we can use labels to get an idea of the number of worlds in which an action supports a subgoal. A technique we experiment with during relaxed plan extraction is preferring actions that support subgoals in more worlds (coverage) – as we may need to include less actions to support subgoals from all the worlds in  $BS_s$ . We do not have to compute cost vectors to use the coverage technique, but it is admittedly myopic because it does not consider the cost of using an action for support. Hence we also experiment with cost vectors for vertices.

**Cost Vectors**: A cost vector  $c_k(v)$  is a set of pairs  $\langle f_i, c_i \rangle$ , where  $f_i$  is a propositional formula over  $F$  and  $c_i$  is a rational number. Every  $c_i$  is an estimate of the cost of reaching  $v$  from all state models  $S_s \in \mathcal{M}(f_i)$ .

Cost propagation on planning graphs, similar to that used in the Sapa planner [Do and Kambhampati, 2003], computes the estimated cost of reaching literals at time points. The cached costs give relaxed plans an estimate of the cost associated with including an action (in terms of the cost incurred to support the preconditions of the chosen action). In using the costs, we face a more general scenario where there may be different costs for every subset of models of  $BS_s$ . Instead of tracking costs for an exponential number of subsets, we partition the models of  $BS_s$  into fixed sets to track cost over (i.e. the elements of the cost vectors  $c_k(v)$ ). The fixed sets are different for every literal, action, and effect, because they are defined with respect to labels. The partitions are with respect to the new worlds that support a vertex at a level. We briefly discuss the procedure for cost and label propagation through graph layers by describing how to find each layer of the graph.

**Initial Layer:** The *LUG* has an initial layer,  $\mathcal{L}_0$ , consisting of every literal that is in a model of  $BS_s$ . In the initial layer, the label  $\ell_0(l)$  of each literal  $l$  represents the states of  $BS_s$  in which  $l$  holds. In the cost vector, we store a cost of zero for the entire group of worlds in which each literal is initially reachable (i.e.  $\langle \ell_0(l), 0 \rangle$ ).

**Action Layer:** Based on the previous literal layer  $\mathcal{L}_k$ , the action layer  $\mathcal{A}_k$  contains all non- $\perp$  labelled causative actions with from the action set  $A$ , plus all literal persistence. The label of the action at level  $k$ , is equivalent to the conjunction of the labels of its execution preconditions. If there are new worlds supporting  $a$  at level  $k$ , we add a formula-cost pair to the cost vector with the formula equal to  $\ell_k(a) \wedge \neg \ell_{k-1}(a)$ .<sup>2</sup>

We then update the cost for each element of the cost vector, where each  $f_i$  is a formula describing the new worlds of  $BS_s$  that came to support  $a$  at a distinct level. We find  $c_i$  by summing the costs of the execution precondition literals, in the worlds described by  $f_i$ . The cost of each literal is determined by covering  $f_i$  with the cost vector of the literal.

For example, we wish to compute the cost of an action  $a$  in a set of worlds described by  $f$ , where  $a$  has a single execution precondition  $l$  with a cost vector  $\{\langle f', 3 \rangle, \langle f'', 5 \rangle\}$ . Say  $f$  represents two states  $s_1, s_2$ ,  $f'$  represents  $s_1, s_3$ , and  $f''$  represents  $s_2, s_4$ . We need both  $f'$  and  $f''$  to cover  $f$  because both cover one state of  $f$ . The cost of  $f$  is then 8 because the cost of the cover is 8.

**Effect Layer:** An effect  $\varphi_j$  is included in  $\mathcal{E}_k$ , when it is reachable in some world of  $BS_s$ , i.e.  $\ell_k(\varphi_j) \neq \perp$ , which only happens when both the associated action and the antecedent are reachable in at least one world. The cost  $c_i$  of world set  $f_i$  of an effect at level  $k$  is found by adding the execution cost of the associated action, the support cost of the action in the worlds of  $f_i$ , and the sum of the cost of the antecedent literals in  $f_i$ .

**Literal Layer:** The literal layer,  $\mathcal{L}_k$ , contains all literals with non- $\perp$  labels. The label of a literal,  $\ell_k(l)$ , depends on  $\mathcal{E}_{k-1}$  and is the disjunction of the labels of each effect that gives the literal. The cost  $c_i$  in a set of worlds  $f_i$  for a literal at level  $k$  is found by covering the worlds  $f_i$  with the union of all formula-cost pairs of effects that support the literal.

**Termination:** The *CLUG* construction stops when two consecutive literal layers are identical.

### Relaxed Plans

The *LUG* and *CLUG* relaxed plan heuristics account for positive world interaction and independence across source states in achieving the goals. In the relaxed plan we support the goal with every state in  $BS_s$ , but in doing so we track which states in  $BS_s$  use which paths in the graph. There may be several paths used to support a subgoal in the worlds of  $BS_s$ , because not one supports all worlds. One challenge in extracting the relaxed plan is in doing the proper label algebra to track what worlds use which paths to support subgoals. Another challenge is in extracting cost-sensitive relaxed plans from the *CLUG*. In the next section, we demonstrate both effectiveness of using cost vectors (cost) as in the

*CLUG* and the size of action’s label (coverage) as in the *LUG* to decide which actions to use to support subgoals.

The *LUG* and *CLUG* relaxed plans are inadmissible, i.e. will not guarantee optimal plans with AO\* search. Admissible heuristics are lower bounds that enable search to find optimal solutions, but most in practice are very ineffective for improving search efficiency. In the next section we demonstrate that although our heuristics are inadmissible they guide our planner toward solutions of comparable quality to a planner that uses admissible heuristics and do so much faster.

## Empirical Comparisons

Our main intent is to evaluate the effectiveness of the *LUG* and *CLUG* relaxed plans in improving the quality of plans generated by *POND*. Additionally, we also compare with two state of the art planners, GPT [Bonet and Geffner, 2000], and MBP [Bertoli *et al.*, 2001]. Even though MBP does not plan with costs, we show the expected cost of MBP’s plans for each problem’s cost model. GPT uses admissible heuristics based on relaxing the problem to full-observability (whereas our relaxation is to no observability while ignoring action mutexes), and MBP uses a belief state’s size as its heuristic merit. For lack of space, our test set up involves a single domain, Medical-Specialist. Each problem had a time out of 20 minutes and a memory limit of 1GB on a 2.8GHz P4 Linux machine.

**Medical-Specialist:** We developed an extension of the medical domain [Weld *et al.*, 1998], where in addition to staining, counting of white blood cells, and medicating, one can go to a specialist for medication and there is no chance of dying – effectively allowing conformant (non-sensing) plans where the specialist medication is used for every disease. We assigned costs as follows:  $c(\text{stain}) = 5$ ,  $c(\text{count\_white\_cells}) = 10$ ,  $c(\text{inspect\_stain}) = X$ ,  $c(\text{analyze\_white\_cell\_count}) = X$ ,  $c(\text{medicate}) = 5$ , and  $c(\text{specialist\_medicate}) = 10$ . We generated ten problems, each with the respective number of diseases (1-10), in two sets where  $X = \{15, 25\}$ .

Our results in the first two columns in Figures 1, 2, and 3 show the expected cost, plan breadth, and total time for two cost models. Relaxed plans based on propagated cost instead of coverage enable *POND* to be more cost-sensitive. Using the cost propagation method, plans tend to branch less than coverage as the cost of sensing increases in order to reduce expected cost. Since MBP is insensitive to cost, the its plans are proportionately costlier as the sensor cost increases. GPT returns better plans, but tends to take significantly more time as the cost of sensing increases; this can be attributed to how the heuristic is computed by relaxing the problem to full-observability. Our heuristics measure the cost of co-achieving the goal from a set of states, whereas GPT takes the max cost of the states.

In summary, the experiments show that the *LUG* heuristics help with scalability and using propagated cost to extract relaxed plans helps find better solutions. We also found that planners not reasoning about action cost can return arbitrarily poor solutions, and planners that use weaker assumptions about uncertainty and cost do not scale as well.

<sup>2</sup>When  $k = 0$  we can say  $\ell_{-1}(a) = \perp$ .

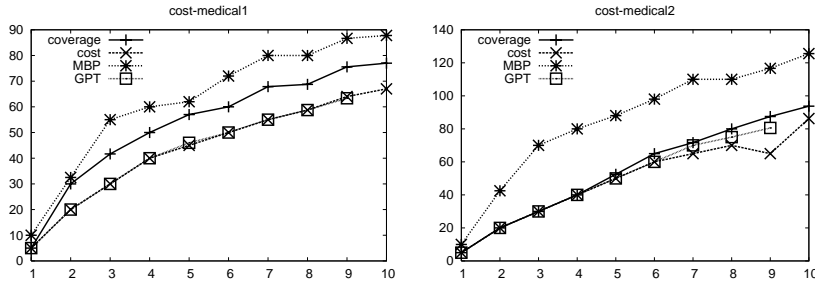


Figure 1: Expected cost results for POND (coverage and cost), MBP, and GPT for Medical-Specialist.

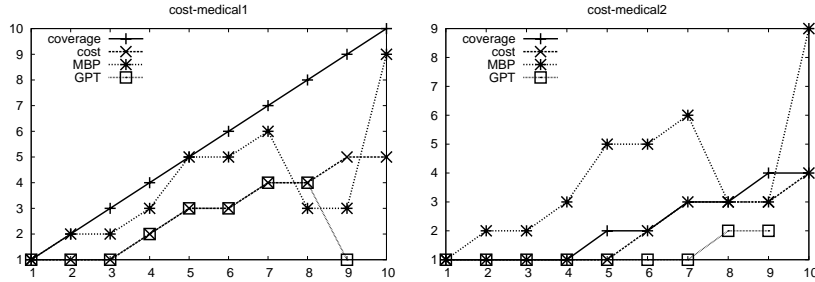


Figure 2: Breadth (# of plan paths) results for POND (coverage and cost), MBP, and GPT for Medical-Specialist.

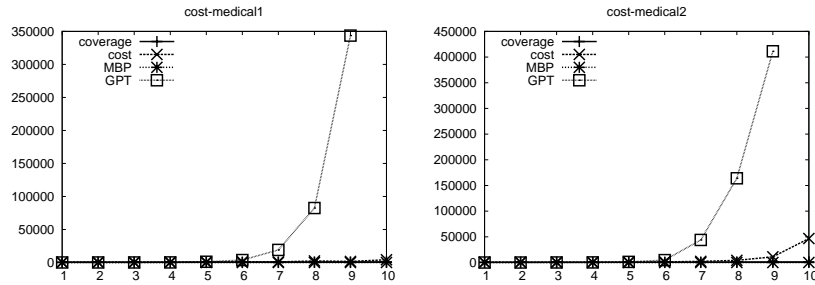


Figure 3: Total Time(ms) results for POND (coverage and cost), MBP, and GPT for Medical-Specialist.

## Conclusion

With our motivation toward conditional planning approaches that can scale like classical planners, but still reason with quality metrics, we have presented a planning graph innovation called the *CLUG*. With this we extract cost-sensitive relaxed plans that are effective in guiding our planner *POND* toward high-quality conditional plans. We have shown with an empirical comparison that our approach improves the quality of conditional plans over conditional planners that do not take cost information into account, and we can out scale previous approaches that consider cost information in a weaker fashion. Given our ability to propagate numeric information on the *LUG*, we are currently adapting these heuristics to handle non uniform probabilities. The extension involves using expected cost in the cost vectors so relaxed plans can select actions such that subgoals are supported with high probability, but low expected cost.

## References

P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in non-deterministic domains under partial observability via symbolic model checking. In *Proceedings of IJCAI'01*, 2001.

B. Bonet and H. Geffner. Planning with incomplete information

as heuristic search in belief space. In *Proceedings of AIPS'02*, 2000.

D. Bryce and S. Kambhampati. Heuristic guidance measures for conformant planning. In *Proceedings of ICAPS'04*, June 2004.

D. Bryce, S. Kambhampati, and D.E. Smith. Planning in belief space with a labelled uncertainty graph. Technical report, AAAI Workshop TR WS-04-08, 2004.

M. Do and S. Kambhampati. Sapa: A scalable multi-objective heuristic metric temporal planner. *JAIR*, 2003.

M. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *JAIR*, 9:1–36, 1998.

O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. In *AAAI/IAAI*, pages 541–548, 1999.

N. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, 1980.

R. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of AIPS'02*, 2002.

J. Rintanen. Complexity of planning with partial observability. In *Proceedings of ICAPS'04*, 2004.

D. Weld, C. Anderson, and D.E. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of AAAI'98*, 1998.