SCALABLE PLANNING UNDER UNCERTAINTY

by

Daniel Bryce

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

ARIZONA STATE UNIVERSITY

May 2007

SCALABLE PLANNING UNDER UNCERTAINTY

by

Daniel Bryce

has been approved

April 2007

Graduate Supervisory Committee:

Subbarao Kambhampati, Chair
Chitta Baral
Goran Konjevod
David E. Smith
David J. Musliner

ACCEPTED BY THE DIVISION OF GRADUATE STUDIES

ABSTRACT

Autonomous agents that act in the real-world can often improve their success by capturing the uncertainty that arises because of their imperfect knowledge and potentially faulty actions. By making plans robust to uncertainty, agents can be prepared to counteract plan failure or act upon information that becomes available during plan execution. Such robust plans are valuable, but are often difficult to construct when uncertainty is high and goal achievement requires many actions. Unfortunately, current approaches to planning under uncertainty are not scalable: they focus on small problems where the challenge is finding an optimal solution.

I study (non-deterministic and probabilistic) conformant and conditional planning in large problems where just finding a feasible solution is the challenge. I develop scalable heuristic search techniques that are inspired both by optimal approaches to planning under uncertainty (based on Markov decision processes) and scalable approaches for classical planning (based on planning graph reachability heuristics). Specifically, I develop measures for the cost of completing partial plans, heuristics that estimate the measures, efficient data-structures and inference techniques to compute the heuristics, and a multi-objective heuristic search algorithm that uses the heuristics. Through extensive empirical evaluation, I show the resulting set of techniques significantly improves the scalability of planning under uncertainty.

This work is dedicated to my loving fiancée.

Without her I would not be the person I am.

ACKNOWLEDGMENTS

This work would not be possible without my esteemed mentors, colleagues, and friends.

My dissertation advisor, Rao Kambhampati, taught me more about automated planning than I ever knew was possible. I thank him for always making time for me and returning my article drafts with an extra pound of red ink. I would know very little about research and all that is not research (but necessary for it) without his guidance.

David E. Smith was unofficially, but effectively, my second advisor and his mentoring was invaluable in completing this work. His vision for planning within complex applications is inspiring. I thank him for knowing the answers and asking me the right questions.

David Musliner helped me gain practical experience with planning technology and provided many insights on government contracting. I thank him for helping me understand how to turn theory into something that actually works and adding to my extensive list of acronyms the most important extension of *LUG* – $GLUG$.

I thank the ASU faculty members Chitta Baral, Goran Konjevod, Seungchan Kim, Huan Liu, Arunabha Sen, and Joohyung Lee that helped me in several capacities, as committee members, instructors, collaborators, and advice givers. I especially thank Chitta Baral for fueling my interest in planning under uncertainty by introducing me to the uncertain view of the world through the eyes of a mobile robot.

I thank many of the current and past members of the Yochan research group, including Terry Zimmerman, Biplav Srivastava, Minh Do, Romeo Sanchez, Will Cushing, Menkes Van den Briel, and J. Benton for their helpful advice, collaboration, questions, and friendship. I especially thank Will Cushing for our many long talks and his help conducting portions of the research contained in this dissertation.

I would also like to thank many of my colleagues who have given me advice and helped further my research in different ways during my time as a graduate student, including Mausam, Eric Hansen, Joerg Hoffmann, Blai Bonet, Hakaan Younes, Shlomo Zilberstein, Craig Boutilier, Hector Palacios, Rune Jensen, Piergiorgio Bertoli, Alessandro Cimmati, Dan Weld, Carmel Domshlak, Stephen Majercik, Nathaniel Hyafil, Fahiem Bacchus, Jussi Rintanen, Nicolas Meuleau, Mark Boddy, Sylvie Thiebaux, Chris Geib, Wheeler Ruml, Tran Cao Son, Satish Kumar Thittamaranahalli, and Nilufer Onder.

Lastly, I would like to thank my family for their encouragement and support in everything leading up to and including completing my dissertation. Most of all I thank my soon to be wife Renée for believing in me and loving me through it all.

This work was supported by a RIACS/NASA summer fellowship, a MCT/NASA summer fellowship, two summer internships at Honeywell Labs, the ARCS Foundation, the ASU College of Engineering, NSF grant IIS-0308139, ONR grant N000140610058, the DARPA Integrated Learning Initiative (through a contract from Lockheed Martin), an IBM faculty award, and student scholarships from ICAPS, AAAI, and IJCAI.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

Plan synthesis is an important, central task of Artificial Intelligence that concerns sequencing actions to achieve a goal. Classical planning models simplify the problem to model situations with perfect state information and deterministic actions. In a number of applications, the classical model is insufficient, and extensions such as durative actions, resources, and uncertainty are needed for faithful representations of an environment. This work concentrates on planning models that capture uncertainty, and specifically, how to scale up the synthesis of plans in such models. Using intuitions from techniques for scaling classical planning and approximate reasoning in probabilistic models, I combine the best of both to improve the synthesis of conformant and conditional plans in non-deterministic and probabilistic models. This chapter will introduce the subsequent chapters by providing motivation for this work, a justification for the methods taken, and a brief summary of each chapter. This chapter will end with a perspective on the impact and implications of this work.

1. Uncertain Environments

When plans are executed by imperfect agents in unfriendly environments, it is important to account for uncertainty or face failure. It is estimated that the $265 million Mars Pathfinder rover spent nearly 40 to 70 percent of its time idle due to plan failure. For example, uncertainty in soil consistency complicates models of power consumption. By planning contingencies, a rover running low on power could branch its plan to take pictures (for some science return) instead of simply failing to finish a long drive (with no science return). Uncertainty is one of the most difficult aspects of plan synthesis, which this work addresses.

Uncertainty can take the form incomplete information, noisy observations, multiple action outcomes, and differing action durations. This work addresses incomplete information and multiple action outcomes. Incomplete information captures the notion that certain facts about the environment that are not always observable. Having multiple action outcomes capture uncertainty in the action's result (e.g., flipping a coin). Reasoning about these types of uncertainty can lead to robust plans.

There are two main techniques for handling uncertainty: be reactive or be proactive. Replanning upon plan failure is a reactive approach that typically uses a single sequence of actions. When the assumptions (under which the plan was constructed) change it becomes necessary to find a new plan, that is, replan. Conditional and conformant plans are proactive approaches that mitigate uncertainties that can contribute to plan failure. With multiple assumptions (capturing the possibility of failure), proactive approaches ensure the plan will succeed for all or most cases. The trade-off between the two extremes is that reactive approaches are less expensive, but suboptimal (and sometimes unsafe), and proactive approaches are of higher utility, but expensive. For example, a rover using a replanning strategy may have to wait for the next day for a new plan (from mission control) if it runs low on power. However, a rover using a conditional planning strategy could observe that its power is low and use a pre-planned contingency to take a picture instead. With proper motivation for pursuing conditional plans, the most pertinent issue is how to automatically synthesize reasonable quality plans for large environments in a reasonable amount of time.

2. Automated Plan Synthesis

Plan synthesis is often posed as a graph search problem, where the graph corresponds to a state transition system. Edges correspond to actions, vertices correspond to states, and plans are paths leading from an initial state to a goal state. Automated planning often assumes propositional (or factored) models that describe states and action in terms of state variables. From the action descriptions it is possible to construct regions of the state transition system. Often the state transition system is quite large (because states are defined by hundreds of variables), making enumeration impossible. Heuristic search algorithms help expand only those regions needed to find a plan. With a sound inference procedure to generate states from action descriptions, the crux of plan synthesis is guiding the heuristic search.

Chapter 2 provides background on plan synthesis in classical planning, as described above. It additionally describes the foundations of reachability heuristics used to guide search. The chapter additionally formalizes several models for planning under uncertainty and a heuristic search algorithm for several classes of plans. The remaining chapters describe generalizations of reachability heuristics for planning under uncertainty and a heuristic search algorithm for an additional class of plans. These chapters are outlined in the next section.

3. Outline of Contributions

Chapters 3 to 8 contain several approaches for improving the scalability of planning under uncertainty. The following list introduces each of these chapters:

- **Planning Graph Heuristics for Belief State Search:** Chapter 3 uses intuitions from classical planning reachability heuristics, described in Chapter 2, to generalize state distance measures to belief state distance measures. Under this generalization, the chapter shows how the belief state distance measure can be estimated using planning graphs.

- **Labeled Planning Graphs:** Chapter 4 presents a new data structure, called a labeled planning graph, to improve the efficiency of computing belief state distance estimates. The labeled planning graph compactly represents and symbolically reasons about sets of planning graphs.

- **Labeled Planning Graphs for Cost-Sensitive Planning:** Chapter 5 presents a generalization of the labeled planning graph to compute estimates of belief state distance when actions have non-uniform cost. The resulting cost labeled planning graph propagates cost information over multiple planning graphs.

- **Labeled Planning Graphs for Actions with Uncertain Effects:** Chapter 6 describes a generalization of the labeled planning graph to incorporate actions with uncertain outcomes. The monte carlo labeled planning graph incorporates monte carlo simulation to avoid reasoning about all uncertain outcomes of actions, instead focussing on the most likely outcomes.

- **State Agnostic Planning Graphs:** Chapter 7 describes a framework, called state agnostic planning graphs, using intuitions from the labeled planning graph. While the labeled planning graph captures a set of planning graphs to estimate belief state distance, the state agnostic planning graph captures arbitrary sets of planning graphs for many planning models. Using state agnostic planning graphs it is possible to build a single planning graph structure for every search node, rather constructing one fore each search node.

- **Multi-Objective Conditional Probabilistic Planning:** Chapter 8 presents a novel approach to conditional probabilistic planning with the multi-objective LAO^* ($MOLAO^*$) search algorithm. $MOLAO^*$ is the first cost-based heuristic search algorithm for finding conditional probabilistic plans. $MOLAO^*$ uses the heuristics developed in the preceding chapters to solve the most general planning problems considered by this work.

4. Impact and Implications

Many important applications require planning under uncertainty, such as web-service composition, work-flow synthesis, and robotics. Scalability, as the goal of this work, is an ever important consideration for making such applications feasible.

Before this work was started in 2002, approaches for planning under uncertainty used relatively small problems where the emphasis was on finding optimal plans. In contrast, most recent work in classical planning has emphasized finding reasonable quality plans for much larger problems, which is justified by practical applications of classical planning (Boddy *et al.*, 2005; Ruml *et al.*, 2005). Because finding optimal plans even in classical planning models is still challenging, it seems limiting to focus on only optimal planning under uncertainty (a much more difficult class of problems). To help impact the applications mentioned above, I focus on scaling planning under uncertainty by removing the restriction of optimality, while still mindful of plan quality.

The main implications of this work rely on its concurrent development with techniques for classical planning and reasoning under uncertainty; it can be seen as a bridge between these two fields. Research on classical planning and Markov decision processes (one of the most popular formalisms for planning under uncertainty) has largely proceeded in isolation. In recent years, many of the similarities between these models have been identified (Boutilier *et al.*, 1999), but few techniques have actively combined the two. This work can be seen as one of the first approaches to combine the fields by taking advantage of scalable techniques from both. In doing so, it provides a clear path to integrate many of the advanced classical planning and reasoning under uncertainty techniques that are to come in the future.

CHAPTER 2

BACKGROUND

This chapter contains the background material and definitions used in later chapters. Specifically, it starts with a presentation of classical planning and reachability heuristics, and then describes assumptions regarding uncertainty and observability that are needed to model agents and their environments. With these assumptions, the chapter defines several planning models, problems expressed in the models, and solutions to the problems. The chapter concludes by describing a heuristic search algorithm for solving planning problems and a planner developed around the search algorithm.

1. Introduction

There are a number of expressive features of planning models, including time, uncertainty, observability, costs/rewards, and resources. Research between the dawn of AI in the 1950's and the 1990's largely explored defining the semantics of plans with each of these features in isolation and in concert. Since the 1990's, research in planning has concentrated significantly more on how to *efficiently* synthesize such expressive plans. Many of the techniques for scalable plan synthesis have roots in classical planning, adapt to deal with more expressive model features. This work describes an adaptation of classical planning reachability heuristics to deal with different types of uncertainty and observability.

We will use variations on the following running example to illustrate how several planning models can address the same problem and how we can derive heuristics for the models:

Example 1 (Planetary Rover). *Ground control needs to plan the daily activities for the rover Conquest. Conquest is currently at location alpha on Mars. The mission scientists have objectives to obtain a soil sample from location alpha, a rock core sample from location beta, and a photo of the lander from the hilltop location gamma. The rover must communicate any data it collects to have the objectives satisfied. The cost of driving from one location to another varies depending on the* *terrain and distance between locations. Actions to collect soil, rock, and image data incur different costs. Mission scientists may not know which locations will have the data they need and the rover's actions may fail, leading to potential sources of uncertainty.*

This work is about the scalable synthesis of plans in uncertain environments, and there are a number of topics upon which it builds. The central concept will be heuristic search in belief space (where the search graph represents how actions make transitions between belief states). This chapter begins by describing how to perform heuristic search in state space for environments without uncertainty. The heuristic search description includes both how the search proceeds and how a specific type of heuristic (based on planning graphs) is computed. Subsequent chapters will address how these ideas are extended to more expressive planning models that capture uncertainty. Then, two sections will cover basic ideas on how uncertainty and observability can be modeled. The following chapter details the semantics of several models for planning under uncertainty and several types of plans that are solutions for problems expressed in the models. The chapter ends with a description of a heuristic search algorithm that will be used for many of the planning models and the $POND$ planner that uses the search algorithm.

2. Classical Planning

This section starts with a brief background on how the classical planning problem is represented and why the problem is difficult, then follows with an introduction to planning graph heuristics for state-based progression (forward-chaining) search.

Background: The classical planning problem is defined as a tuple (P, A, I, G), where P is a set of propositions, A is a set of actions, I is a set of initial state propositions, and G is a set of goal propositions. This description of classical planning uses many representational assumptions (described below) consistent with the STRIPS language (Fikes and Nilsson, 1971). While STRIPS is only one of many choices for action representation, it is very simple and most other action languages can be compiled down to STRIPS (Nebel, 2000). In STRIPS a state s is a proper subset of the propositions P, where every proposition $p \in s$ is said to be true (or to hold) in the state s. Any proposition $p \notin s$ is false in s. The set of states S is the power set of P, such that $S = 2^P$. The initial state s_I is specified by a set of propositions $I \subseteq P$ known to be true (the false propositions are inferred by the closed world assumption) and the goal is a set of propositions $G \subseteq P$ that must be made true in a state s for s to be a goal state. Each action $a \in A$ is described by an execution cost $c(a)$, a set of propositions $\rho_e(a)$ for execution preconditions, and an effect $(\varepsilon^+(a), \varepsilon^-(a))$ where $\varepsilon^+(a)$ is a set of propositions that it causes to become true and $\varepsilon^-(a)$ is a set of propositions it causes to become false. An action a is applicable $appl(a, s)$ to a state s if each precondition proposition holds in the state, $\rho_e(a) \subseteq s$. The successor state s' is the result of executing an applicable action a in state s, where $s' = exec(a, s) = s \backslash \varepsilon^-(a) \cup \varepsilon^+(a)$. A sequence of actions $\{a_1, ..., a_n\}$, executed in state s, results in a state s', where $s' = exec(a_n, exec(a_{n-1}, ... \; exec(a_1, s) \; ...))$ and each action is executable in the appropriate state. A valid plan is a sequence of actions that can be executed from s_I and results in a goal state. The sum of the costs of the actions $\sum_{i=1}^{n} c(a_i)$ is the cost of the plan.

STRIPS planning problems can be described with the planning domain description language (PDDL) (McDermott, 1998). Figure 1 is a PDDL formulation of the rover problem for classical planning.[1] On the left is a domain description and on the right is a problem instance. The domain description uses predicates and action schemas with free variables to abstractly define a planning domain. The problem instance defines objects, an initial state, and a goal. Through a process called *grounding*, objects defined in the problem description are used to instantiate predicates and action schemas. Grounding involves using every combination of objects to replace free variables

[1]In the following, I will refer to the `alpha`, `beta`, and `gamma` locations in the text, but use the related symbols α, β, and γ to simplify the illustrations.

```
(define (domain rovers_classical)
  (:requirements :strips :typing)
  (:types location data)
  (:predicates
          (at ?x - location)
          (avail ?d - data ?x - location)
          (comm ?d - data)
          (have ?d - data))
  (:action drive
   :parameters (?x ?y - location)
   :precondition (at ?x)
   :effect (and (at ?y) (not (at ?x))))
  (:action commun
   :parameters (?d - data)
   :precondition (have ?d)
   :effect (comm ?d))
  (:action sample
   :parameters (?d - data ?x - location)
   :precondition (and (at ?x) (avail ?d ?x))
   :effect (have ?d))
)
```

```
(define (problem rovers_classical1)
  (:domain rovers_classical)
  (:objects
       soil image rock - data
       alpha beta gamma - location)
  (:init (at alpha)
         (avail soil alpha)
         (avail rock beta)
         (avail image gamma))
  (:goal (and (comm soil)
              (comm image)
              (comm rock)))
)
```

Figure 1. PDDL description of classical planning formulation of the rover problem.

in predicates to obtain propositions, and in action schemas to obtain ground actions. The problem instance in Figure 1 denotes that the initial state is:

$$s_I = \{\texttt{at(alpha), avail(soil, alpha),}$$
$$\texttt{avail(rock, beta), avail(image, gamma)}\}$$

and that the goal is:

$$G = \{\texttt{comm(soil), comm(image), comm(rock)}\}.$$

The domain description in Figure 1 lists three action schemas for driving between two locations, communicating data, and obtaining data by sampling. For example, the `drive` action schema can be instantiated with the `alpha` and `beta` location objects to obtain the ground action `drive(alpha, beta)` where its precondition is $\{\texttt{at(alpha)}\}$ and it causes $\{\texttt{at(beta)}\}$ to become true and $\{\texttt{at(alpha)}\}$ to become false:

$$\rho_e(\texttt{drive(alpha, beta)}) = \{\texttt{at(alpha)}\}$$

$$\varepsilon^+(\texttt{drive(alpha, beta)}) = \{\texttt{at(beta)}\}$$

$$\varepsilon^-(\texttt{drive(alpha, beta)}) = \{\texttt{at(alpha)}\}$$

Executing `drive(alpha, beta)` from the initial state results in the state:

$$\begin{aligned} s' &= exec(\texttt{drive(alpha, beta)}, s_I) \\ &= \{\texttt{at(beta), avail(soil, alpha), avail(rock, beta),} \\ &\quad \texttt{avail(image, gamma)}\}, \end{aligned}$$

because `at(beta)` becomes true and `at(alpha)` becomes false. A valid plan for the problem in Figure 1 is the following sequence of actions:

```
( sample(soil, alpha),   commun(soil),
  drive(alpha, beta),    sample(rock, beta),
  commun(rock),          drive(beta, gamma),
  sample(image, gamma),  commun(image))
```

Reachability Heuristics: Classical planning can be viewed as finding a path from an initial state to a goal state in a state transition graph. This view suggests a simple algorithm that constructs the state transition graph and uses a shortest path algorithm to find a plan in $O(n \log n)$ time. However, practical problems have a very large number of states n. In the example, there are a total of 18 propositions, giving $n = 2^{18} = 2.6 \times 10^5$ states (which may be feasible in the above algorithm). By making the problem more realistic, adding 17 more locations (a total of 20), 12 additional types of data (a total of 15), and another rover, there are 420 propositions and $n = 2^{420} = 2.7 \times 10^{126}$ states.[2] With approximately 10^{87} particles in the universe, explicitly representing and searching a state transition graph of this size is impractical.

Instead of an explicit graph representation, it is possible to use a search algorithm and a propositional representation to construct regions of the state transition graph, as needed. However,

[2]As we will see, the search space can also be infinite (e.g., in probabilistic planning in belief space).

in the worst case, it is possible to still construct the entire transition graph. Heuristic search algorithms, such as A* search, can "intelligently" search for plans and hopefully avoid visiting large regions of the transition graph. The critical concern of such heuristic search algorithms is the design of a good heuristic.

To illustrate heuristic search for plans, consider the most popular search formulation, progression (a.k.a. forward-chaining). The search creates a projection tree rooted at the initial state s_I by applying actions to leaf nodes (representing states) to generate child nodes. Each path from the root to a leaf node corresponds to a plan prefix, and expanding a leaf node generates all single step extensions of the prefix. A heuristic estimates the "goodness" of each leaf node, and in classical planning this can be done by measuring the cost to *reach* a goal state (hence the terminology *reachability heuristics*). With the heuristic estimate, search can focus effort on expanding the most promising leaf nodes.

For instance, consider the empty plan prefix (starting at the initial state s_I) in our example. Possible extensions of the plan prefix include driving to other locations, or sampling soil at the current location. While each of these extensions contain actions relevant to supporting the goals, they have different completion costs. If the rover drives to another location, then at some point it will need to come back and obtain the soil sample. It would be better to obtain the soil sample now to avoid extra driving later. A reachability heuristic should be able to measure this distinction.

Exact & Approximate Reachability Information: An obvious way to compute exact reachability information is to compute the full projection tree rooted at the initial state. The projection tree for the example is depicted in Figure 2a. The projection is represented as states in dark boxes connected via edges for actions. The propositions holding in each state are listed (except for the `avail` propositions, which are in all states).[3] Within this tree, the exact reachability cost for each

[3]It is common to remove static propositions from state and action descriptions because they do not change value.

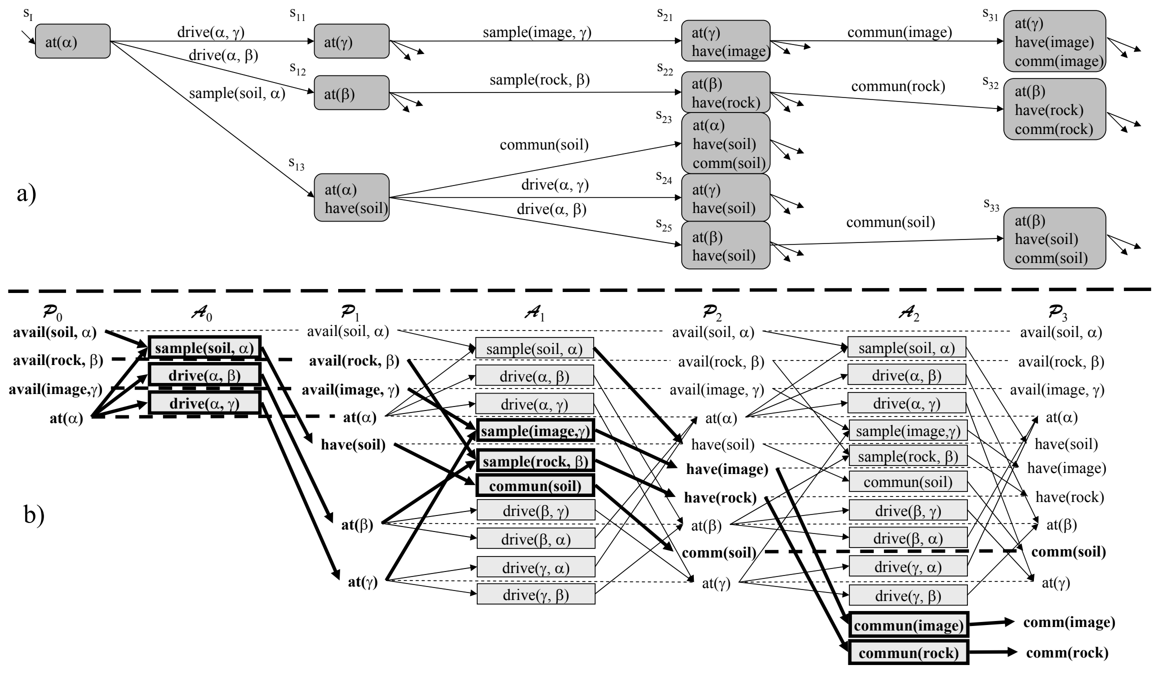


Figure 2. Progression search graph (a) and planning graph (b) for rovers problem.

node is the minimal length path to reach a state satisfying the goal. For example, the cost of reaching `at(beta)` is 1 and the cost of reaching `have(rock)` is 2. It is easy to see that access to such exact reachability information can guide the search well.

Expecting exact reachability information is impractical as it is no cheaper than the cost of solving the original problem! Instead, research on classical planning explores more efficient ways of computing reachability information approximately. Of particular interest are "optimistic" (or lower-bound) approximations as they can provide the basis for admissible heuristics. It turns out that the planning graph data structure suits this purpose quite well. Figure 2b shows the planning graph for the rover problem in juxtaposition with the exact projection tree. The planning graph is a layered graph structure with alternating action and proposition layers (with the former shown in rectangles). There are edges between layers: an action has its preconditions in the previous layer and its positive effects in the next layer. For instance, the `sample(soil, alpha)` action, which is applicable at every level, has incoming edges from its precondition propositions `avail(soil, alpha)` and `at(alpha)`, and an outgoing edge for its positive effect `have(soil)`. We ignore negative effects (Hoffmann and Nebel, 2001) because action preconditions are strictly positive. In addition to the normal domain actions, the planning graph also uses "persistence" actions (shown by the dotted lines) which can be seen as noops which take and give back specific propositions. It is easy to see that unlike the projection tree, the planning graph structure can be computed in polynomial time.

There is an obvious structural relationship between the planning graphs and projection trees: the planning graph seems to correspond to an envelope over the projection tree. In particular, the action layers seem to correspond to the union of all actions at the corresponding depth in the projection tree, and the proposition layers correspond to the union of all the states at that depth (with the states being treated as "sets" of propositions). The envelope analogy turns out to be more than syntactic—the proposition and action layers can be viewed as defining the upper bounds on the feasible actions and states in a certain formal sense. Specifically, every legal state s at depth d in the projection tree must be a subset of the proposition layer at level d in the planning graph. The converse however does not hold. For instance, $\mathcal{P}_1$ contains the propositions `at(beta)` and `have(soil)`, but they do not appear together in any state at depth 1 of the search graph. In other words, the planning graph data structure is providing optimistic reachability estimates.

The planning graph can be viewed as the exact projection tree for a certain relaxation of the domain. The relaxation involves ignoring the interactions between and within action effects during graph expansion. Two axioms capture these interactions and distinguish state expansion in the projection tree versus proposition layer expansion in the planning graph. In the projection tree, the first axiom expresses that the state (set of propositions) resulting from applying action a_1 is *exclusive* of the state resulting from applying a_2. For example, at depth 1 of the projection tree applying `sample(soil, alpha)` to state s_1 makes `have(soil)` true in only state s_{13}; `have(soil)` is not true in the other states. The second axiom states that the effect propositions of each action must hold together in the resulting state (i.e., they are *coupled*). Applying `drive(alpha, beta)` to state s_1 makes `at(gamma)` true and `at(alpha)` false in state s_{11}; without coupling the effects, the state would allow both `at(gamma)` and `at(alpha)` to be true. Expanding a proposition layer also involves applying several actions, but with modifications to the two axioms above. The first modified axiom states that the set of propositions resulting from applying a_1 is *independent* of the propositions resulting from applying a_2, meaning that they are neither exclusive, nor coupled. Consider how both `at(gamma)` and `have(soil)` appear in the first proposition layer $\mathcal{P}_1$ of the planning graph (suggesting that the state {`at(gamma)`, `have(soil)`} is reachable at depth 1). The second modified axiom states that the effect propositions of each action are also *independent*. For example, the first proposition layer $\mathcal{P}_1$ contains both `at(gamma)` and `at(alpha)` via the independence within the effect of `drive(alpha, gamma)`; the assumption allows ignorance of how `drive(alpha, gamma)` makes `at(alpha)` false. The projection tree maintains state barriers and couplings between effect propositions, where the planning graph removes both constraints. With an intuitive structural interpretation, the formal definitions of the planning graph and the heuristics that it encodes follow quite easily.

Planning Graphs: I start by formalizing planning graphs and follow with a description of several planning graph based reachability heuristics. Traditionally, progression search uses a different planning graph to compute the reachability heuristic for each state s. A planning graph $PG(s, A)$, constructed for the state s (referred to as the projected state) and the action set A is a leveled graph, captured by layers of vertices $(\mathcal{P}_0(s), \mathcal{A}_0(s), \mathcal{P}_1(s), \mathcal{A}_1(s), ..., \mathcal{A}_k(s), \mathcal{P}_{k+1}(s))$, where each level i consists of a proposition layer $\mathcal{P}_i(s)$ and an action layer $\mathcal{A}_i(s)$. In the following, I simplify the notation for a planning graph to $PG(s)$, assuming that the entire set of actions A is always used.

The notation for action layers $\mathcal{A}_i$ and proposition layers $\mathcal{P}_i$ also assumes that the state s is implicit.

The first proposition layer, $\mathcal{P}_0$, is defined as the set of propositions in the state s. An action layer $\mathcal{A}_i$ consists of all actions that have all of their precondition propositions in $\mathcal{P}_i$. A proposition layer $\mathcal{P}_i$, $i > 0$, is the set all propositions given by the positive effect[4] of an action in $\mathcal{A}_{i-1}$. It is common to use implicit actions for proposition persistence (a.k.a. noop actions) to ensure that propositions in $\mathcal{P}_{i-1}$ persist to $\mathcal{P}_i$. A noop action a_p for proposition p is defined as $\rho_e(a_p) = \varepsilon^+(a_p) = p$.

Planning graph construction can continue until one of the following conditions holds: (i) the graph has *leveled off* (i.e., two subsequent proposition layers are identical), or (ii) the goal is reachable (i.e., every goal proposition is present in a proposition layer). In Figure 2a the planning graph has all of the goal propositions $\{$`comm(soil), comm(rock), comm(image)`$\}$ in $\mathcal{P}_3$ and will level off at $\mathcal{P}_4$. It is also possible to truncate planning graph construction at any level. If the goal is not reachable before truncation, then the number of levels is still a lower bound on the number of steps to reach the goal. However, if the goal is not reachable before the graph has leveled off, then the goal is not reachable and there is no plan.

Heuristic Estimates of Plan Cost: Planning graph heuristics are used to estimate the plan cost to transition between two states, a source and a destination state. The source state is always the state that defines $\mathcal{P}_0$, and the destination state is one of potentially many goal states.

There are two fundamental types of planning graph heuristics, level-based and relaxed plan heuristics (Nguyen *et al.*, 2002). Most level-based heuristics compute a cost $c(s, g)$ to reach each proposition $g \in G$ from the state s and then numerically aggregate the costs, via maximization ($\max_{g \in G} c(s, g)$) or summation ($\sum_{g \in G} c(s, g)$). The heuristics are level-based because the cost

[4]The reason that actions do not contribute their negative effects to proposition layers (which contain only positive propositions) is a syntactic convenience of using STRIPS. Since action preconditions and the goal are defined only by positive propositions, it is not necessary to reason about reachable negative propositions. In general, actions languages (such as ADL (Pednault, 1994)) allow negative propositions in preconditions and goals, requiring the planning graph to maintain "literal" layers that record the all reachable values of propositions (Koehler *et al.*, 1997).

Heuristic	G	G_1	G_2
Max	3	2	1
Sum	8	3	2
Relaxed Plan	8	2	2
True Cost	8	2	2

Table 1. Heuristic estimates and true cost to achieve each goal from the initial state.

of each proposition p is determined by the index of the first proposition layer in which it appears, $c(s,p) = min_{i:p \in \mathcal{P}_i} i$. For instance, the goal proposition `comm(soil)` first appears in $\mathcal{P}_2$, meaning it has a cost of 2. If a proposition does not appear in any proposition layer of a leveled off planning graph, then its cost is ∞. Otherwise, if the planning graph has k levels but has not leveled off, the cost of such missing propositions is at least k. Unlike level-based heuristics that relate level to cost, relaxed plans identify the actions needed to causally support all goals (while ignoring negative interactions). I explore both types of heuristics in detail here, but later chapters focus exclusively on relaxed plans.

In order to illustrate the different heuristics (see Table 1), I will use three goals for the rover problem: G = $\{$`comm(soil), comm(image), comm(rock)`$\}$ (the original goal), G_1 = $\{$`at(beta), have(rock)`$\}$, and G_2 = $\{$`at(beta), have(soil)`$\}$. Table 1 lists the cost estimates made by different heuristics for each goal. The table also lists the true optimal plan cost for each goal. The following discussion uses these goals to explain the properties of each heuristic.

Level-based Heuristics: The level-based heuristics make a strong assumption about the cost of reaching a set of propositions: namely, that achievement cost is related to the level where propositions first appear. Depending on how one assesses the cost of the set of propositions, additional assumptions are made.

Using maximization to aggregate proposition costs assumes that the actions used to achieve the set of propositions will positively interact. For example, with G_1 it is possible to achieve at(beta) in $\mathcal{P}_1$ defining c(s_I, at(beta)) = 1, and achieve have(rock) in $\mathcal{P}_2$, defining c(s_I, have(rock)) = 2. Taking the maximum of the costs max(1, 2) = 2 to achieve each proposition avoids counting the cost of drive(alpha, beta) twice. The max heuristic for reaching the goal G from s_I is $\max(c(s_I,$have(soil)$), c(s_I,$ have(rock)$), c(s_I,$have(image)$)) = \max(2, 3, 3) = 3$.

Using a summation to aggregate proposition costs assumes that the sub-plans to achieve the set of propositions will be fully *independent*. Independence captures the notion that actions taken to achieve one goal will neither aid nor prevent achievement of another goal. For example, with G_2 it is possible to achieve at(beta) in $\mathcal{P}_1$, defining c(s_I, at(beta)) = 1, and achieve have(soil) in $\mathcal{P}_1$, defining c(s_I, have(soil)) = 1. Taking the summation of the costs 1 + 1 = 2 accurately measures the plan cost as 2 (because the required actions were independent), whereas taking a maximization would under-estimate the plan cost as 1. The sum heuristic for reaching the original goal G is 8 because $c(s_I,$have(soil)$) + c(s_I,$have(rock)$) + c(s_I,$have(image)$)) = 2 + 3 + 3 = 8$. In practice the sum heuristic usually outperforms the max heuristic, but gives up admissibility.

The main problem with level-based heuristics is that they assume that the proposition layer index is equal to the cost of achieving a proposition. Because planning graphs optimistically allow multiple parallel actions per step, using level to define cost can be misleading. Consider a goal proposition that first appears in $\mathcal{P}_2$: its cost is 2. In reality the proposition may be supported by a single action with 100 precondition propositions, where each proposition must be supported by a different action. Thus, a plan to support the goal would contain 101 actions, but a level-based heuristic estimates its cost as 2. One can overcome this limitation by using a relaxed plan heuristic.

Relaxed Plan Heuristics: Many of the problems with level-based heuristics came from ignoring how multiple actions per level execute in parallel. The reachability heuristic should better estimate

```
RPExtract($PG(s), G$)
1:  Let $n$ be the index of the last level of $PG(s)$
2:  **for all** $p \in G \cap \mathcal{P}_n$ **do** /* Initialize Goals */
3:      $\mathcal{P}_n^{RP} \leftarrow \mathcal{P}_n^{RP} \cup p$
4:  **end for**
5:  **for** $i = n...1$ **do**
6:      **for all** $p \in \mathcal{P}_i^{RP}$ **do** /* Find Supporting Actions */
7:          Find $a \in \mathcal{A}_{i-1}$ such that $p \in \varepsilon^+(a)$
8:          $\mathcal{A}_{i-1}^{RP} \leftarrow \mathcal{A}_{i-1}^{RP} \cup a$
9:      **end for**
10:     **for all** $a \in \mathcal{A}_{i-1}^{RP}, p \in \rho_e(a)$ **do** /* Insert Preconditions */
11:         $\mathcal{P}_{i-1}^{RP} \leftarrow \mathcal{P}_{i-1}^{RP} \cup p$
12:     **end for**
13: **end for**
14: **return** $(\mathcal{P}_0^{RP}, \mathcal{A}_0^{RP}, \mathcal{P}_1^{RP}, ..., \mathcal{A}_{n-1}^{RP}, \mathcal{P}_n^{RP})$
```

Figure 3. Relaxed Plan Extraction Algorithm.

the number of actions in a plan. Through a simple back-chaining algorithm (Figure 3) called *relaxed plan extraction*, it is possible to identify the actions in each level that are needed to support the goals or other actions. Relaxed plans are subgraphs $(\mathcal{P}_0^{RP}, \mathcal{A}_0^{RP}, \mathcal{P}_1^{RP}, ..., \mathcal{A}_{n-1}^{RP}, \mathcal{P}_n^{RP})$ of the planning graph, where each layer corresponds to a set of vertices. Where appropriate, I represent relaxed plans by their action layers (omitting noop actions and empty action layers) and omitting all proposition layers. A relaxed plan satisfies the following properties: (i) every proposition $p \in \mathcal{P}_i^{RP}$, $i > 0$, in the relaxed plan is supported by an action $a \in \mathcal{A}_{i-1}^{RP}$ in the relaxed plan, and (ii) every action $a \in \mathcal{A}_i^{RP}$ in the relaxed plan has its preconditions $\rho_e(a) \subseteq \mathcal{P}_i^{RP}$ in the relaxed plan. A relaxed plan captures the causal chains involved in supporting the goals, but ignores how actions may conflict. For example, a relaxed plan extracted from $PG(s_I)$ may contain both `drive(alpha, beta)` and `drive(alpha, gamma)` in $\mathcal{A}_0^{RP}$ because they both help support the goal, despite conflicting.

Figure 3 lists the algorithm used to extract relaxed plans. Lines 2-4 initialize the relaxed plan with the goal propositions. Lines 5-13 are the main extraction algorithm that starts at the last level of the planning graph n and proceeds to level 1. Lines 6-9 find an action to support each proposition in a level. Line 7 is the most important critical step in the algorithm. It is common to prefer noop actions for supporting a proposition (if possible) because the relaxed plan is likely to include fewer extraneous actions. For instance, a proposition may support actions in multiple levels of the relaxed plan; by supporting the proposition at the earliest possible level, it can persist to later levels. It also possible to select actions based on other criterion, as will be seen in later chapters. Lines 10-12 insert the preconditions of chosen actions into the relaxed plan. The algorithm ends by returning the relaxed plan, which is used to compute the heuristic, which is the total number of non noop actions in the action layers.

The relaxed plan to support the goal G from s_I is depicted in Figure 2b in bold. Each of the goal propositions is supported by a chosen action, and each of the actions has its preconditions supported. There are a total of eight actions in the relaxed plan, each with unit cost, so the heuristic is 8.

The advantage of relaxed plans is that they capture both positive interaction and independence of sub-plans used to achieve the goals, rather than assuming one or the other. The goals G_1 and G_2 have the respective relaxed plans:

$$\begin{aligned}\mathcal{A}_0^{RP(G_1)} &= \{\texttt{drive(alpha, beta)}\},\\ \mathcal{A}_1^{RP(G_1)} &= \{\texttt{sample(rock, beta)}\}\end{aligned}$$

and

$$\mathcal{A}_0^{RP(G_2)} = \{\texttt{sample(soil, alpha), drive(alpha, beta)}\}.$$

In the above, the relaxed plan heuristic is equivalent to the max heuristic for the positively interacting goal propositions in G_1, and it is equivalent to the sum heuristic for the independent goal propositions in G_2. Relaxed plans measure both action independence and positive interaction, making them a compromise between the max and sum heuristics (which measure just one or the other).

Heuristic search in classical planning deals with estimating the distance between states. However, when there is uncertainty, the techniques described above may not work as well. The next chapter will evaluate this claim with respect to incomplete state information. To describe planning under uncertainty, a model of uncertainty is first needed, described next.

3. Modeling Uncertainty

When mission control from the rover example has less than perfect knowledge of the rover's environment they may want to describe their uncertainty. The classical planning model does not allow uncertainty, but it can be extended. There are two types of uncertainty addressed in this work: non-deterministic (possibilistic) and stochastic (probabilistic). The former refers to uncertainty by a set possibilities, and the latter refers to a probability distribution over possibilities. For example, in the rover problem, there may be uncertainty about which location will provide a good soil sample. The model can describe this uncertainty with propositions `avail(soil, alpha)`, `avail(soil, beta)`, and `avail(soil, gamma)`. A non-deterministic model may assert that exactly one of {`avail(soil, alpha)`, `avail(soil, beta)`, `avail(soil, gamma)` } holds. The probabilistic model may assert that `avail(soil, alpha)` holds with probability 0.4, `avail(soil, beta)` holds with probability 0.5 and `avail(soil, gamma)` holds with probability 0.1. In both cases, uncertainty is modeled in terms of a set of *possible worlds*. Possible world semantics describes the set of joint outcomes to a set of uncertain events as possible worlds. Probabilistic uncertainty adds weights to possible worlds, where non-deterministic uncertainty does not. In the example above, in each possible world the soil is at a different location.

The meaning behind probabilistic uncertainty should be clear, but there can be many interpretations of the non-deterministic uncertainty:

1. Each possibility is equally likely (i.e., there is an implicit uniform probability distribution over possibilities).

2. There is a probability distribution over possibilities, but it is unknown.

3. There is a probability distribution over possibilities, but it is ignored.

Each of these interpretations is appropriate under different circumstances, and it is the job of the domain modeler to decide which applies.

Some works (Eiter and Lukasiewicz, 2003; Trevizan *et al.*, 2007) have combined the two types of uncertainty to allow for a non-deterministic uncertainty over probability distributions or a probability distribution over non-deterministic choices. While these combined models could have practical use in situations where some probability distributions are known and others are not, I concentrate on exclusively non-deterministic and probabilistic models of uncertainty.

There also exists a third type of uncertainty that will not be addressed, namely qualitative uncertainty. Seen as a compromise between non-deterministic and stochastic uncertainty, qualitative uncertainty assigns possibilities to a small set of qualitative assessments such as: “unlikely”, “possible”, or “highly likely”.

The following sections will give definitions for planning with uncertainty, pointing out where differences due to non-deterministic and stochastic uncertainty arise.

4. Observability

If, in the rover example, mission control is uncertain about where there is soil, then perhaps the rover is able to obtain some information to help rule out uncertainty. When a model includes uncertainty, observability captures the degree to which the outcome of an uncertain event is known. There are three basic types of observability:

1. Full: It is possible to uniquely and correctly determine the outcome of an uncertain event.

2. Partial: It is possible to characterize the posterior possibility (probability) of an outcome, but not necessarily to the extent that the outcome is uniquely determined. Partial observations may also be noisy, meaning that information about the event is potentially faulty.

3. None: It is not possible to characterize the outcome of an uncertain event aside from its prior possibility (probability).

There are several important consequences of assuming a particular type of observability. The first is that partial observability contains full and no observability as special extreme cases. Second, assuming partial and no observability require reasoning about hidden outcomes, where with full observability this is not required. To see the differences, consider the rover example from the previous section where there is uncertainty about which location has soil. Assume also that the action to sample the soil may or may not work (its effect is uncertain). Under full observability, the assumption is that the rover is pre-programmed with knowledge of which location has soil and when the rover uses the `sample` action, it will know if it worked. With no observability, the rover will not know which location has the soil and it will not know when the `sample` action succeeds. With partial observability, it may be that the rover does not know which location has the soil, but it will know if the `sample` action works. If the rover has a noisy sensor, then it may think that the `sample` action worked when it did not, and vice-versa.

In planning, there are specific models for each type of uncertainty and observability. These models formulate actions under various assumptions that reflect the uncertainty and observability assumptions listed above.

5. Planning under Uncertainty

Given the types of uncertainty and observability described in the previous sections, there are a number of different combinations that give rise to different planning models. This section will detail a general formulation that captures all of these combinations and defines the restrictions on the general formulation that define each specific model. In the general model, it becomes necessary to model how incomplete state information leads to belief states (by characterizing possible hidden states), and how actions can have uncertain effects and provide observations about hidden states. Also, with a more general state and action model, plans can have very different structure, taking the form of digraphs versus sequences.

In the following, I define the shared features of several planning models. Each planning problem uses a subset of these features in its definition. From the planning models and problem definitions, I will define the semantics of plans (solutions). The following section builds upon this section by detailing a search algorithm that can be used to synthesize many of the plans.

5.1. Model Features. Building upon the classical planning formulation from earlier, it becomes necessary to describe belief states and actions with uncertain effects and observations.

Belief States: When the state is hidden, it is characterized by a belief state b, which represents all possible or probable current states. In the remainder of this work, states in a belief state may also be referred to as possible worlds. Depending on the model of uncertainty, belief states are defined differently.

A non-deterministic belief state b is a *set of possible states* $b \subseteq S$. The set of all non-deterministic belief states B is the powerset of all states, which is the power set of all propositions, such that $B = 2^{2^P} = 2^S$. A non-deterministic belief state also describes a function $b : S \rightarrow \{0, 1\}$, where $b(s) = 1$ if $s \in b$ and $b(s) = 0$ if $s \notin b$.

A probabilistic belief state b is a *probability distribution over states*, describing a function $b : S \rightarrow [0, 1]$, such that $\sum_{s \in S} b(s) = 1.0$. While every state is involved in the probability distribution, many are often assigned zero probability. To maintain consistency with non-deterministic belief states, those states with non-zero probability are referred to as states in the belief state $s \in b$ if $b(s) > 0$. Probabilistic belief states can be seen as the weighted generalization of non-deterministic belief states.

Actions: In classical planning it is often sufficient to describe actions by their execution precondition and positive and negative effects. With uncertainty, it is convenient to describe actions that have multiple outcomes with context-dependent (conditional) effects and observations. Redefining actions, an action $a \in A$ is a tuple $(\rho_e(a), \Phi(a), \mathcal{O}(a))$, where $\rho_e(a)$ is an enabling precondition, $\Phi(a)$ is a set of causative outcomes, and $\mathcal{O}(a)$ is a set of observables. The enabling precondition $\rho_e(a)$ is a conjunctive set of propositions that determines the states in which an action is applicable. An action a is applicable $appl(a, s)$ to state s if $\rho_e(a) \subseteq s$, and it is applicable $appl(a, b)$ to a belief state b if for each state $s \in b$ the action is applicable.[5]

Each causative outcome $\Phi_i(a) \in \Phi(a)$ is a set of conditional effects. Each conditional effect $\varphi_{ij}(a) \in \Phi_i(a)$ is of the form $\rho_{ij}(a) \rightarrow (\varepsilon^+_{ij}(a), \varepsilon^-_{ij}(a))$ where both the antecedent (secondary precondition) $\rho_{ij}(a)$, the positive consequent $\varepsilon^+_{ij}(a)$, and the negative consequent $\varepsilon^-_{ij}(a)$ are a conjunctive set of propositions. Actions are assumed to be consistent, meaning that for each $\Phi_i(a) \in \Phi(a)$ each pair of conditional effects j and j' have consequents such that $\varepsilon^+_{ij}(a) \cap \varepsilon^-_{ij'}(a) = \emptyset$ if there is a state s where both may execute (i.e., $\rho_{ij}(a) \subseteq s$ and $\rho_{ij'}(a) \subseteq s$). In other words, no two conditional effects of the same outcome can have consequents that disagree on a proposition if both effects are applicable. In non-deterministic models, each outcome has a weight $w_i(a) = 1$. In

[5]While it may seem like a strong restriction that actions must have their execution precondition satisfied in every state of a belief state for the action to be applicable, it is possible to move execution preconditions into conditional effects. By doing so, the action may be applicable in more belief states, but the effects will only change certain states that satisfy the conditional effect preconditions. Another alternative, that is not explored in this work, is to allow uncertainty in action applicability, which makes plan executability uncertain.

probabilistic models, there is a weight $0 < w_i(a) \leq 1$ indicating the probability of each outcome i being realized, such that $\sum_i w_i(a) = 1$. This representation of effects follows the 1ND normal form presented by Rintanen (2003).

As outlined in the probabilistic PDDL (PPDDL) standard (Younes and Littman, 2004), it is possible to use the effects of every action to derive a state transition function $T(s, a, s')$ that defines a possibility or probability that executing a in state s will result in state s'. Executing action a in state s will result in a single state s' for each outcome $\Phi_i(a)$:

$$s' = exec(\Phi_i(a), s) = s \cup \left(\bigcup_{j:\rho_{ij} \subseteq s} \varepsilon^+_{ij}(a) \right) \setminus \left(\bigcup_{j:\rho_{ij} \subseteq s} \varepsilon^-_{ij}(a) \right)$$

In non-deterministic models, this defines the possibility of transitioning from state s to s' by executing a as $T(s, a, s') = 1$ if there exists an outcome $\Phi_i(a)$ where $s' = exec(\Phi_i(a), s)$, and $T(s, a, s') = 0$, otherwise. In probabilistic models $T(s, a, s')$ is the sum of the weight of each outcome where $s' = exec(\Phi_i(a), s)$, such that:

$$T(s, a, s') = \textstyle\sum_{i:s'=exec(\Phi_i(a),s)} w_i(a)$$

Executing action a in belief state b, denoted $exec(a, b) = b_a$, defines the successor belief state $b_a(s')$ in the following manner. In non-deterministic models $b_a(s') = \max_{s \in b} b(s)T(s, a, s')$, and in probabilistic models $b_a(s') = \sum_{s \in b} b(s)T(s, a, s')$.

Each observable $\mathcal{O}_k(a) \in \mathcal{O}(a)$ is described by a set of propositions (i.e., $\mathcal{O}_k(a) \subseteq P$), and in probabilistic models it has an associated pair of probabilities $(w^+_k(a), w^-_k(a))$, indicating its reliability in terms of a respective true positive and true negative probability. Observations in non-deterministic models are assumed to be reliable (i.e., $w^+_k(a) = w^-_k(a) = 1$).

If a state s satisfies the observable $\mathcal{O}_k(a)$ (i.e., $\mathcal{O}_k(a) \subseteq s$), then with probability $w^+_k(a)$ the observable $\mathcal{O}_k(a)$ is reliably observed true in state s. Similarly, $w^-_k(a)$ is the probability that $\mathcal{O}_k(a)$ is reliably observed false in state s when $\mathcal{O}_k(a) \not\subseteq s$. Let $\Omega(a) : 2^{\mathcal{O}_k(a)}$ denote the set of possible observations after executing action a, where each observation $o \in \Omega(a)$ is a set of observables observed to be true (assuming those observables $\mathcal{O}_k(a) \notin o$ are observed false). In non-deterministic models, define the possibility of each observation $o \in \Omega(a)$ with the observation function $O(a, s, o)$, where $O(a, s, o) = 1$ if $\forall_{\mathcal{O}_k(a) \in o} \mathcal{O}_k(a) \subseteq s$ and $\forall_{\mathcal{O}_k(a) \notin o} \mathcal{O}_k(a) \not\subseteq s$, and $O(a, s, o) = 0$ otherwise. In probabilistic models, define the probability of each observation $o \in \Omega(a)$ with the observation function $O(a, s, o) = \prod_k Pr(\mathcal{O}_k(a)|s)$, where

$$Pr(\mathcal{O}_k(a)|s) = \begin{cases} w_k^+(a) & : \mathcal{O}_k(a) \in o, \mathcal{O}_k(a) \subseteq s \\ w_k^-(a) & : \mathcal{O}_k(a) \notin o, \mathcal{O}_k(a) \not\subseteq s \\ 1 - w_k^-(a) & : \mathcal{O}_k(a) \in o, \mathcal{O}_k(a) \not\subseteq s \\ 1 - w_k^+(a) & : \mathcal{O}_k(a) \notin o, \mathcal{O}_k(a) \subseteq s \end{cases}$$

The observation function helps define the belief state b_a^o that results from assimilating observation o while in belief state b_a, such that $b_a^o = assim(o, b_a)$, $b_a^o(s) = \alpha b_a(s) O(a, s, o)$, and $\alpha = \sum_{s \in b_a} b_a(s) O(a, s, o)$ is a normalization constant. The probability of receiving observation o of belief state b_a is equivalent to α.

Combining action execution and observation assimilation, define $filter((a : o), b) = assim(o, exec(a, b)) = b_a^o$. Define the belief state b' reached by a sequence of actions and observations $(a_0 : o_0, a_1 : o_1, ..., a_m : o_m)$ as $b' = filter((a_0 : o_0, a_1 : o_1, ...a_m : o_m), b) = filter((a_m : o_m), ...filter((a_1 : o_1), filter((a_0 : o_0), b))...)$.

5.2. Planning Models. With the definitions of planning model features, it is possible to define a generic planning model (to which various restrictions define models of interest):

Definition 1 (Generic Planning Model)**.** *The generic planning model is a tuple $GP = (P, A, b_I, G)$, where P is a set of propositions, A is a set of actions, b_I is a description of the initial belief state, and G is a set of goal propositions.*

The restrictions on observability are defined as follows:

- No Observability (NO): Each action a has a single observable, $\mathcal{O}(a) = \{\emptyset\}$.
- Full Observability (FO): Each action a has an *implicit* observable for every proposition, $\mathcal{O}(a) = \{\mathcal{O}_k(a) = \{p\} | p \in P\}$.
- Partial Observability (PO): Each action a has a possibly non empty set of observables.
  - Perfect (PPO): Each action a has weight $w^+_k(a) = w^-_k(a) = 1$ for each observable $\mathcal{O}_k(a) \in \mathcal{O}(a)$.
  - Noisy (NPO): There is an action a that has weight $0 \leq w^+_k(a) < 1$ or $0 \leq w^-_k(a) < 1$ for some observable $\mathcal{O}_k(a) \in \mathcal{O}(a)$.

The restrictions on action effects are as follows:

- Deterministic Actions (DA): Each action a has a single outcome, $\Phi(a) = \{\Phi_0(a)\}$.
- Non-Deterministic Actions (NDA): Each outcome $\Phi_i(a)$ of each action a has no weight $w_i(a)$.
- Probabilistic Actions (PA): Each outcome $\Phi_i(a)$ of each action a has a weight $w_i(a)$, such that $\sum_i w_i(a) = 1$.

The restrictions on the initial state are as follows:

- Deterministic Initial State (DI): There is a single initial state s, such that $b_I(s) = 1$, and $\forall s' \neq s, b_I(s') = 0$.
- Non-Deterministic Initial State (NDI): There is a non-deterministic initial belief state b_I.
- Probabilistic Initial State (PI): There is a probabilistic initial belief state b_I.

Figure 4 summarizes the models of interest in terms of their assumptions on uncertainty and observability. It shows a portion of the belief space constructed for an example rover problem in each of the models. Figure 5 lists a problem description for the rover example. The problem description contains several features, such as probabilistic effects, noisy observations, and a probabilistic initial belief state. To describe each of the models, various features of the example will be ignored.

In the example, the probabilistic initial belief state is comprised of three states:

$$s_\alpha = \{\texttt{at(alpha), avail(soil, alpha)}\}$$

$$s_\beta = \{\texttt{at(alpha), avail(soil, beta)}\}$$

$$s_\gamma = \{\texttt{at(alpha), avail(soil, gamma)}\}$$

such that $b_I = \{0.4 : s_\alpha, 0.5 : s_\beta, 0.1 : s_\gamma\}$. The goal is to achieve `comm(soil)`, and valid solutions must do so with at least 0.5 probability. The action that will be illustrated is the `sample(soil, alpha)` action, described as:

$\rho_e(\texttt{sample(soil, alpha)}) = \{\texttt{at(alpha)}\}$

$\varphi_{00}(\texttt{sample(soil, alpha)}) = \{\{\texttt{avail(soil, alpha)}\} \rightarrow (\{\texttt{have(soil)}\}, \{\})\}$

$w_0(\texttt{sample(soil, alpha)}) = 0.9$

$\varphi_{10}(\texttt{sample(soil, alpha)}) = \{\}$

$w_1(\texttt{sample(soil, alpha)}) = 0.1$

$\mathcal{O}0(\texttt{sample(soil, alpha)}) = \{\texttt{have(soil)}\}$

$w_0^+(\texttt{sample(soil, alpha)}) = 0.9$

$w_0^-(\texttt{sample(soil, alpha)}) = 0.7$

The classical planning model is the most restrictive of the models considered in this work.

Definition 2 (Classical Planning Model)**.** *The classical planning model is defined as the generic planning model with no observability, deterministic actions, and a deterministic initial state. [(NO)*

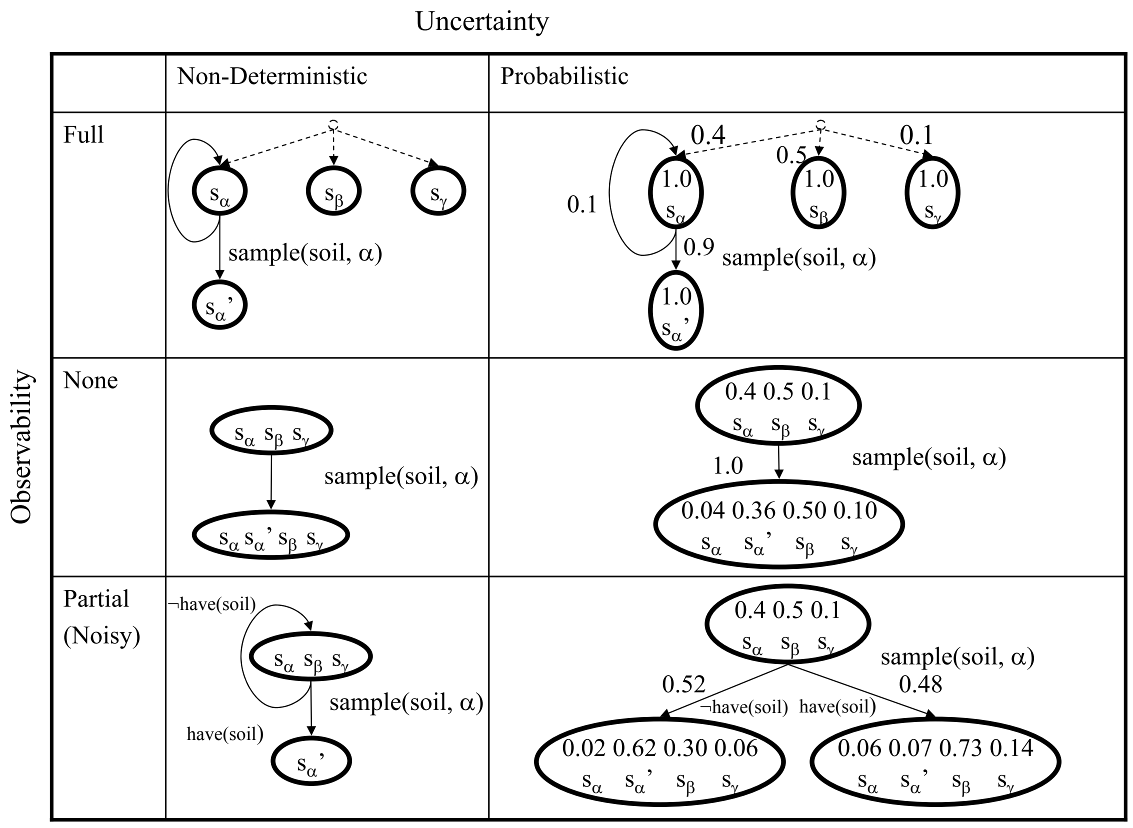


Figure 4. Planning Models.

+ (DA) + (DI)]

Each action has only one outcome and there is a single state in the initial belief state, meaning every belief state reached by executing a sequence of actions will have only one state. Because there are no observations (or need for observations), plans are sequential. The only differences between this definition of classical planning and the definition from this chapter's section on classical planning is a more general action formalism and the notion of belief states. While the actions can contain conditional effects, it is possible to transform these actions in STRIPS-style actions by moving the antecedents of the conditional effects into execution preconditions.

The next model adds a minimum requirement for modeling uncertainty, namely uncertain

```
(define (domain rovers_stochastic)
  (:requirements :strips :typing)
  (:types location data)
  (:predicates
          (at ?x - location)
          (avail ?d - data ?x - location)
          (comm ?d - data)
          (have ?d - data))
  (:action drive
   :parameters (?x ?y - location)
   :precondition (at ?x)
   :effect (and (at ?y) (not (at ?x))))
  (:action commun
   :parameters (?d - data)
   :precondition (and)
   :effect (when (have ?d)
                (probabilistic 0.8 (comm ?d))))
  (:action sample
   :parameters (?d - data ?x - location)
   :precondition (at ?x)
   :effect (when (avail ?d ?x)
                (probabilistic 0.9 (have ?d))))
  :observation ( ((have soil) 0.9 0.7) )
)
```

```
(define (problem rovers_stochastic1)
  (:domain rovers_stochastic)
  (:objects
     soil image rock - data
     alpha beta gamma - location)
  (:init (at alpha)
        (probabilistic 0.4 (avail soil alpha)
                       0.5 (avail soil beta)
                       0.1 (avail soil gamma))
  (:goal (comm soil) 0.5)
)
```

Figure 5. PDDL description of stochastic planning formulation of rover problem.

(non-deterministic) action effects with full observability.

Definition 3 (Fully-Observable Non-Deterministic Conditional Planning). *Fully-observable non-deterministic conditional planning is defined as the generic planning model with full observations, non-deterministic actions, and a non-deterministic initial state. [(FO) + (NDA) + (NDI)]*

Like the classical planning model, every belief state reachable by some action sequence contains a single state in this model. With full observability it is possible to uniquely determine the state. Thus, applying an action with uncertain effects to a belief state may result in several successor belief states, each with a single state that is consistent with a different outcome of the action. When problems have initial state uncertainty, there is assumed to be a dummy initial action that gives each possible initial state as an effect. Figure 4 graphically depicts this dummy action with the dashed lines, leading to each of the initial states. The figure also shows the result of applying the `sample(soil, alpha)` action to the belief state with s_α. Since this model is non-deterministic, assume that the effect of the action is non-deterministic rather than probabilistic. Since the antecedent of the conditional effect $\{$`avail(soil, alpha)`$\}$ is true in s_α, if outcome $\Phi_0($`avail(soil, alpha)`$)$ occurs, the resulting belief state will contain the state:

$s'_\alpha = \{$`at(alpha), avail(soil, alpha), have(soil)`$\}$,

and if outcome $\Phi_1($`avail(soil, alpha)`$)$ occurs, the resulting belief state will contain the state s_α. While not depicted in the figure, applying the same action to the other belief states will result in the same respective belief states because regardless of the outcome, there will be no effect.

It is possible to extend the non-deterministic model to include probabilistic uncertainty in the following model.

Definition 4 (Fully-Observable Probabilistic Conditional Planning)**.** *The fully-observable probabilistic conditional planning model is defined as the generic planning model with full observations, probabilistic actions, and a probabilistic initial state. [(FO) + (PA) + (PI)]*

Fully-observable probabilistic conditional planning is similar to the non-deterministic version, but uses probabilistic actions. As depicted in Figure 4 in the upper right entry of the table, the edges denoting action outcomes are weighted with their probability. Similar to a Markov chain, it is possible to assess the probability of making transitions between belief states. For example, applying `sample(soil, alpha)` to the belief state $\{s_\alpha\}$ results in the belief state $\{s'_\alpha\}$ with probability 0.4(0.9) = 0.36. If the first application did not work, then applying a second time will increase the probability to 0.4(0.9) + 0.4(1.0-0.9)(0.9) = 0.396.

By assuming full observability, or by using the classical planning model, the space of belief states is exactly the same size as the space of states. However, with partial or no observability, the size of the space of belief states is much greater. In non-deterministic models, there are $2^{|S|}$ belief states, but in the probabilistic model there are an infinite number (for all probability distributions over S). The first model that requires belief states (that contain several states) is the non-deterministic conformant planning model.

Definition 5 (Non-Deterministic Conformant Planning)**.** *The non-deterministic conformant planning model is defined as the generic planning model with no observability, and one or both of non-deterministic actions and a non-deterministic initial state. [(NO) + (NDA) + (NDI), (NO) + (DA) + (NDI), (NO) + (NDA) + (DI)]*

The non-deterministic conformant planning model is based on the notion that planning starts with an uncertain belief state and/or an uncertain belief state is reached by applying an action with an uncertain outcome (which cannot be observed). Figure 4 shows the result of applying the `sample(soil, alpha)` action to b_I (ignoring probabilities), resulting in a belief state where s'_α is a possible state in addition to the states already in b_I. Without observations, as in the fully observable model, it is not possible to differentiate the states.

By adding probabilities, it is possible to place a probability distribution over the states in a belief state, and, while the exact state is not known, there is some information about which states are most likely. However, the belief state space becomes much larger with probabilistic information. Repeated applications of `sample(soil, alpha)` will not change the belief state in the non-deterministic model because no new states are possible. Yet, in the probabilistic model, repeated applications of an action may not change the possible states, but it will change the probability distribution (meaning the belief state is different).

Definition 6 (Probabilistic Conformant Planning)**.** *The probabilistic conformant planning model is defined as the generic planning model with no observability, and one or both of probabilistic actions and a probabilistic initial state. [(NO) + (PA) + (PI), (NO) + (DA) + (PI), (NO) + (PA) + (DI)]*

Like the non-deterministic conformant planning model, uncertain effects or an uncertain initial state combine with no observability to require the agent to maintain a belief state. As previously noted, the probabilistic model provides a probability distribution over the outcome of each action and the states in each belief state. Applying `sample(soil, alpha)` to b_I results in the belief state $\{0.04 : s_\alpha, 0.36 : s'_\alpha, 0.50 : s_\beta, 0.10 : s_\gamma\}$. Applying the action again shifts the probability distribution over the same states to the belief state $\{0.004 : s_\alpha, 0.396 : s'_\alpha, 0.50 : s_\beta, 0.10 : s_\gamma\}$.

With observations it is possible to disambiguate the states in a belief state and branch the plan for each possible observation, like the fully observable model. However, because observations are partial, assimilating the observation may result in a belief state with several states. If the observation is perfect, the resulting belief state contains states that agree on the observation. However, if the observation is noisy, then the resulting belief state may contain states that do not agree with the observation because the observation is potentially wrong. Observations in the non-deterministic model are assumed to be perfect and in the probabilistic model they can be noisy.

Definition 7 (Non-Deterministic Conditional Planning)**.** *The non-deterministic conditional planning model is defined as the generic planning model with partial observations, non-deterministic actions, and a non-deterministic initial state. [(PPO) + (NDA) + (NDI)]*

Non-deterministic conditional planning is the most general of the non-deterministic planning models. The initial state can be uncertain, actions can have multiple uncertain outcomes, and the observations are partial. Unlike the fully-observable non-deterministic planning model, it may never be possible to know the outcome of an action. Consider the belief states resulting from applying `sample(soil, alpha)` to b_I. There is one belief state containing the single state s'_α because `have(soil)` is observed true in the state, and another belief state b_I containing all states where `have(soil)` is observed false.

With noisy observations and probabilistic dynamics, the belief states resulting from applying an action with observations will reflect increased probability for states agreeing with the respective observations. However, it may not be possible to completely disambiguate the states.

Definition 8 (Probabilistic Conditional Planning)**.** *The probabilistic conditional planning model is defined as the generic planning model with partial observations, probabilistic actions, and a probabilistic initial state. [(PPO) + (PA) + (PI), (NPO) + (PA) + (PI)]*

The probabilistic conditional planning model is the most general of all the planning models considered in this work. In addition to adding probabilities to the non-deterministic conditional planning model, it adds the possibility of noisy observations. Figure 4 shows the belief states resulting from applying `sample(soil, alpha)` to b_I. The first belief state contains states inconsistent with the `have(soil)` observation, but with low probability, and the second belief state contains a state inconsistent with ¬`have(soil)`, but with low probability. Since the observation is noisy, there is a small probability that it is wrong and some states cannot be completely ruled out from the belief state. The probability of getting each observation is equivalent to the sum of the products of the probabilities of each state and the probability of getting the observation in the state.

Each of the planning models admits different types of solutions to the problems described in the model. The next subsection contains definitions for several types of plans, which are different solutions. The remainder of this work will assume certain solutions are required for each of the models. For example, non-deterministic models will require strong plans (i.e., plans that guarantee goal achievement), and probabilistic models will not (requiring a probabilistic guarantee of goal achievement). Where appropriate, the assumptions on plans and models will be clarified.

5.3. Types of Plans. There are many types of plans that are solutions to the previously defined planning models. Each plan is characterized by the following features: branching, cycles, and goal achievement. Branching corresponds to conditional plans where it is possible to use a different sub-plan for each observation history (defined below). Cycles require branching to support repeated execution of a sub-plan until an observation leads to a different sub-plan. Goal achievement characterizes the degree to which a plan guarantees a goal state is reached upon executing the plan.

Plan traces (hypothetical executions) help define the semantics of a plan π. A plan trace $\tilde{\pi}$ is a sequence $(s_0, a_0, o_0, ..., s_{k-1}, a_{k-1}, o_{k-1}, s_k)$ that starts in an initial state $s_0 \in b_I$, contains k actions and observations, and ends in a state s_k. Each action a_i, $i > 0$, is said to be *conditioned* on the observation o_{i-1}. The first action a_0 is conditioned on the initial belief state (the initial observation). A plan trace is *valid* if for each $0 \leq i < k$: $o_i \in \Omega(a_i)$ and $s_{i+1} \in filter((a_i : o_i), s_i)$. A plan trace satisfies the goal if it is valid and s_k is a goal state. The observation history of a plan trace is the sequence of observations $(o_0, ..., o_{k-1})$.

A plan π is a directed graph $\pi = (V^\pi, E^\pi)$ that summarizes a set of plan traces, such that each plan trace corresponds to a path in the graph. Let $b_I \in V^\pi$ denote a vertex for the initial belief state, $\pi(b)$ denote the action to execute in belief state $b \in V^\pi$, and $(b, b') \in E^\pi$ denote an edge between belief states b and b'. The graph is structured such that there exists a plan trace $(s_0, a_0, o_0, ..., s_{k-1}, a_{k-1}, o_{k-1}, s_k)$ if $\forall\ i : 0 \leq i < k$ where $b_{i+1} = filter((a_0 : o_0, ..., a_i : o_i), b_I)$:

- $b_I = b_0$
- $b_i, b_{i+1} \in V^\pi$, $(b_i, b_{i+1}) \in E^\pi$
- $s_i \in b_i$, $a_i = \pi(b_i)$, $s_{i+1} \in b_{i+1}$
- $\pi(b_k) = \perp$

In other words, a plan trace is in a plan if each state in the plan trace is in the belief state reached by the same action and observation history.

Model	Plan Criterion
Classical	Strong, Acyclic, Sequential
Fully-Observable Non-Deterministic Conditional	Strong, Cyclic, Branching
Non-Deterministic Conformant	Strong, Acyclic, Sequential
Non-Deterministic Conditional	Strong, Acyclic, Branching
Fully-Observable Probabilistic Conditional	Strong/Weak, Cyclic, Branching
Probabilistic Conformant	Strong/Weak, Acyclic, Sequential
Probabilistic Conditional	Strong/Weak, Cyclic, Branching

Table 2. Plan success criterion for the models described in this work.

A plan is *strong* if every plan trace satisfies the goal ($G \subseteq s_k$), otherwise it is *weak*. *Cyclic* plans have an infinite number of plan traces, and *acyclic* plans have a finite number of plan traces. Since there are a finite number of states, observations, and actions, the only way for a plan to have an infinite number of plan traces is through cycles. Each plan trace enters the cycle a different number of times. A *sequential* plan requires that every plan trace has the same observation history (implying they share the same sequence of actions because actions are conditioned on observations), and a *branching* (conditional) plan has at least two plan traces with different observation histories.

In probabilistic models it is possible to quantify the probability of satisfying the goal in weak plans. Define the probability of a plan trace as the probability of the observation history and the starting state $Pr(\tilde{\pi}) = \sum_{i=0}^{k-1} O(s_i, a_i, o_i) b_I(s_0)$. The probability of satisfying the goal in a probabilistic plan is the sum of the probabilities of the valid plan traces that satisfy the goal.

Table 2 lists the solution criterion for each of the models considered in this work. The non-deterministic conditional model is also required to be acyclic because strong cyclic plans for planning with partial observability are beyond the scope of this work; consult Bertoli *et al.* (2001) for more details. The probabilistic models can be weak, but specify a τ bounding the minimum probability of goal satisfaction in acceptable solutions.

6. AO^* Search Algorithm

Recall that from the section on classical planning, progression search is a method to construct a transition graph rooted at the initial state. With uncertainty, the transition graph (also known as the *belief state space*) is rooted at the initial belief state. Belief states are vertices and actions are directed edges, similar to the definition of plans from the previous section. (In fact, the belief state space graph summarizes many partial plans, and the current plan is induced by choosing a best action $\pi(b)$ from the applicable actions for each belief state b.) Because applying an action may result in several successor belief states (one for each observation) a new search algorithm is needed. A* search can be used to find paths in the transition graph, but AO* is needed to find subgraphs. I will use AO* to find directed acyclic subgraphs in all types of conformant planning and non-deterministic conditional planning. Probabilistic conditional planning will require a variation of the LAO* algorithm to find cyclic subgraphs. I will not consider finding cyclic solutions for non-deterministic problems.

AO* constructs the transition graph by maintaining and expanding a current partial solution. Initially, the belief state space contains a single leaf vertex for the initial belief state. Expanding the plan involves expanding every leaf vertex of a partial plan. Expanding a vertex (corresponding to belief state b) adds every belief state b_a^o to the belief state space that is reachable by executing action a and receiving observation o in b. Each successor belief state b_a^o of belief state b is connected by a directed edge (b, b_a^o) labeled by (a, o). Search expands only the leaf vertices of the current partial plan.

Search maintains and expand the current best partial solution by carefully choosing the best action $\pi(b)$ for each belief state b in the partial solution. It is possible to quantify the value of each action a in a belief state b with a $q(b, a)$ function and to quantify the value of a belief state with a $J(b)$ function, such that in non-deterministic models:

```
AO⋆()
1:  i = 0
2:  repeat
3:     Z = ExpandPlan($(b_I, i^{++})$)
4:     Z' = AddAncestors(Z)
5:     Update(Z')
6:  until |Z| = 0
7:  if solved($b_I$) then
8:     return $\pi$
9:  else
10:    return $\emptyset$
11: end if

ExpandPlan($b, i$)
1:  if expanded($b$) $< i$ then
2:     expanded($b$) = $i$
3:     for $(b, b^o_{\pi(b)}) \in E$ do
4:        Z' = ExpandPlan($b^o_{\pi(b)}$)
5:        Z = Z ∪ Z'
6:     end for
7:  else
8:     expanded($b$) = $i$
9:     GenerateSuccessors(b)
10:    $Z = Z \cup b$
11: end if
12: return Z
```

Figure 6. AO⋆ Search Algorithm.

$$q(b,a) = c(a) + \frac{1}{|\Omega(a)|}\sum_{o\in\Omega(a)} J(b^o_a),$$

and in probabilistic models:

$$q(b,a) = c(a) + \sum_{s\in S}\sum_{s'\in S} b(s)T(s,a,s') \sum_{o\in\Omega(a)} \alpha O(a,s',o)J(b^o_a),$$

denotes the cost of executing action a plus the expected cost of the successors of a. The cost of a belief state b is the minimum cost of actions executable in b:

$$J(b) = \min_{a\in A} q(b,a)$$

The best action to perform in b is the minimum cost action:

$$\pi(b) = \operatorname*{argmin}_{a\in A} q(b,a).$$

```
GenerateSuccessors($b$)
 1: if satisfyGoal($b, \tau$) then
 2:     $J(b) = 0$
 3:     solved($b$) = true
 4:     terminal($b$) = true
 5: else if $|A(b)| = 0$ then
 6:     $J(b) = 0$
 7:     solved($b$) = false
 8:     terminal($b$) = true
 9: else
10:     for $a \in A, o \in \Omega(a)$ do
11:         $V = V \cup \{b_a^o\}$
12:         $E = E \cup \{(b, b_a^o)\}$
13:         expanded($b_a^o$) = 0
14:         $J(b_a^o) = h(b_a^o, G)$
15:     end for
16: end if
```

Figure 7. `Generate Successors` Algorithm.

Leaf vertices have no successors with which to compute the q functions, so their $J(b)$ function is initialized with an estimate of its value. The $J(b)$ function represents the estimated expected cost of a plan starting at belief state b and executing $\pi(b)$.

The `AO*` algorithm, listed in Figure 6 iteratively constructs the belief space graph $\mathcal{G} = (V, E)$ rooted at b_I. The algorithm involves three iterated steps: expand the current partial plan π with the `ExpandPlan` routine (line 3), collect the ancestors Z' of newly expanded vertices Z (line 4), and compute the current best partial plan (line 5). The algorithm finishes when it expands no new vertices. If b_I is not marked as solved, then there is no plan, otherwise the plan is π.

In the following I briefly describe the sub-routines used by AO^*. The `ExpandPlan` routine recursively walks the current plan to find unexpanded vertices (lines 1-6). Upon finding a vertex to expand (lines 7-11), it generates all successors of the vertex with the `GenerateSuccessors` routine. Generating successors involves determining if the vertex is a belief state that satisfies the goal and marking it as solved and terminal (lines 1-4). A belief state satisfies the goal in non-

```
AddAncestors(Z)
1: Z' = Z
2: while $\exists b$ s.t. $(b, b^o_{\pi(b)}) \in E$ and $b^o_{\pi(b)} \in$ Z do
3:     Z' = Z' $\cup b$
4: end while
5: return Z'

Update(Z)
1: while $|$Z$| > 0$ do
2:     Remove $b \in$ Z s.t. $\neg\exists b' \in$ Z where $(b, b') \in E$
3:     if terminal($b$) == false then
4:         Backup ($b$)
5:     end if
6: end while

Backup($b$)
1: for $a \in A(b)$ do
2:     Compute $q(b, a)$
3: end for
4: $J(b) = \max_{a \in A(b)} q(b, a)$
5: $\pi(b) = \operatorname{argmax}_{a \in A(b)} q(b, a)$
6: solved($b$) = $\wedge_{o \in \Omega(\pi(b))}$solved($b^o_{\pi(b)}$)
```

Figure 8. AO⋆ Subroutines.

deterministic models if $\forall_{s \in b} G \subseteq s$, and in probabilistic models if $\sum_{s \in b: G \subseteq s} b(s) \geq \tau$. Else, if there are no applicable actions, the belief state is marked unsolved and terminal (lies 5-8). Else, each successor is added to the search graph, and initialized with a heuristic value (lines 9-15). The heuristic h is perhaps the most crucial aspect of the algorithm, which this work is dedicated to defining. A good heuristic will help the search identify which belief states have a low cost to reach the goal, forcing their inclusion in the best partial solution.

After expanding the current plan, `ExpandPlan` returns the set of expanded vertices Z. In order for `Update` (Figure 8) to find the current best partial plan, given the new vertices, it calls `AddAncestors` to add to Z' every ancestor vertex of a vertex in Z whose best action leads a vertex in Z'. The resulting set of vertices consists of every vertex whose value (and best action) can change after `Update`. The `Update` routine iteratively removes vertices from Z that have no descendent in Z and calls `Backup` on non-terminal vertices until no vertices remain in Z. The `Backup` routine computes the value of a vertex, sets its best action, and determines if it is solved (i.e., there is a feasible sub-plan starting at the vertex). The reason `Update` chooses vertices with no descendent in Z is to ensure each vertex has its value updated with the updated values of its children.

AO^* can often avoid computing the entire belief space graph $\mathcal{G}$, leading to significant savings in many problems. By initializing vertices with a heuristic estimate of their value it is possible to ignore vertices that have a consistently highest value. For example in `Backup`, if there exists an action whose q-value is always less than the alternative actions, then the best action will never be set to one of the alternatives. Further, because the alternative actions are never considered best, `ExpandSolution` will never expand the successors.

6.1. AO* Search Example. To illustrate AO*, consider the following variation on the non-deterministic conditional planning version of the rovers problem. Assume that all action effects are deterministic, unlike the previous versions of the rover example. AO* finds only acyclic plans and including non-deterministic actions in conditional planning may violate this restriction (see Figure 4). Both the `sample` and `commun` actions will have a single outcome with a single conditional effect, and the `sample` action will provide a perfect observation of `have(soil)`.

Figure 9 shows the belief state space expanded to find a strong conditional plan for the problem. The belief states in bold and the dashed edges in bold correspond to the plan. All other belief states and edges, are generated as part of the search, but not used in the plan. Each edge is annotated with an action and observation pair, and terminal nodes are denoted by double circles. Each belief state contains a set of states (listed in Figure 10). I will assume that the value of each belief state (to the right of each node) is initialized with an exact heuristic estimate (i.e., the optimal

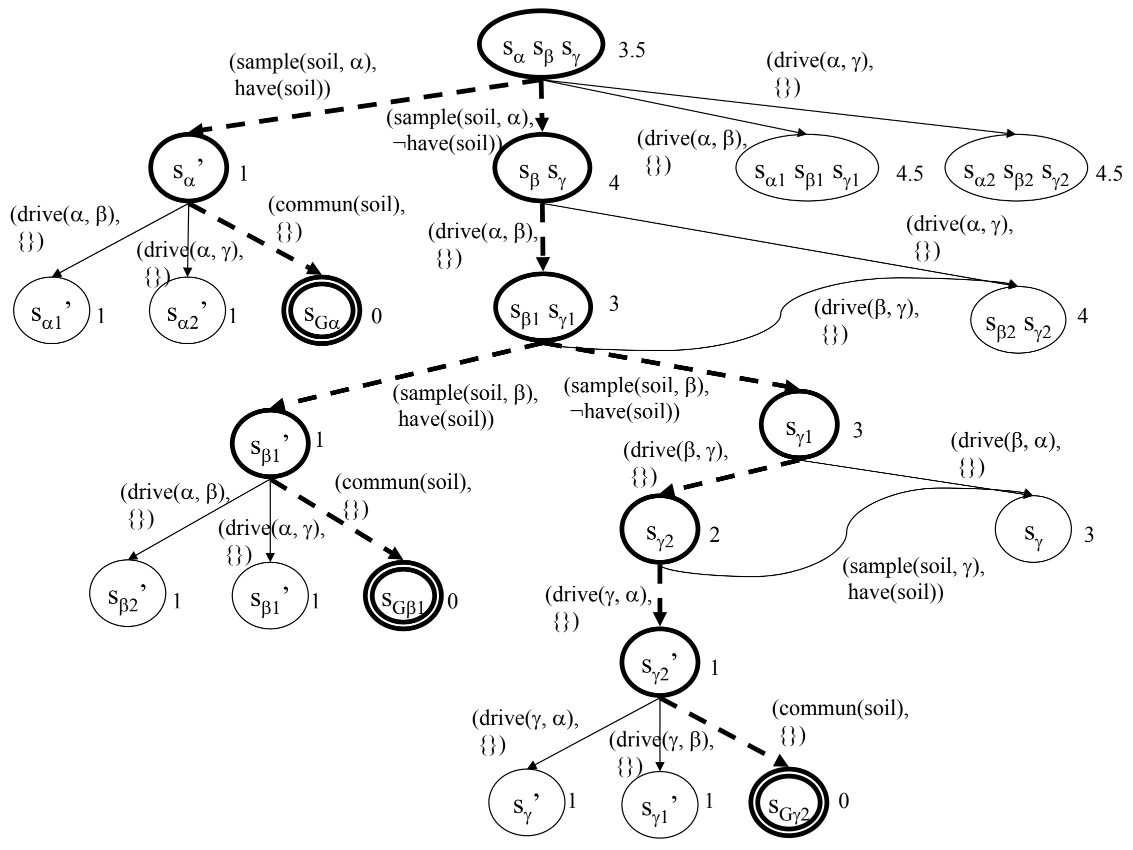


Figure 9. AO* search example.

average cost to reach a terminal node). Thus, the search can find an optimal feasible plan with the minimal number of search node expansions. In the following, I will describe the steps taken by AO* to find the plan.

Initially, the belief state space consists of the root belief state b_I (at the top of the figure). AO* generates the children of b_I by applying the actions: `sample(soil, alpha)`, `drive(alpha, beta)`, `drive(alpha, gamma)`. There are two nodes corresponding to the `sample(soil, alpha)` action because each is consistent with a different observation of `have(soil)`. AO* computes the q-values of applying each action to the initial belief state b_I, and selects the `sample(soil, alpha)` action as the best action because its q-value of 3.5 is

$$\begin{array}{lcl}
s_{\alpha} & = & \{\texttt{at(alpha)}, \texttt{avail(soil, alpha)}\} \\
s_{\alpha 1} & = & \{\texttt{at(beta)}, \texttt{avail(soil, alpha)}\} \\
s_{\alpha 2} & = & \{\texttt{at(gamma)}, \texttt{avail(soil, alpha)}\} \\
s'_{\alpha} & = & s_{\alpha} \cup \{\texttt{have(soil)}\} \\
s'_{\alpha 1} & = & s_{\alpha 1} \cup \{\texttt{have(soil)}\} \\
s'_{\alpha 2} & = & s_{\alpha 2} \cup \{\texttt{have(soil)}\} \\
s_{G\alpha} & = & s'_{\alpha} \cup \{\texttt{comm(soil)}\} \\
s_{\beta} & = & \{\texttt{at(alpha)}, \texttt{avail(soil, beta)}\} \\
s_{\beta 1} & = & \{\texttt{at(beta)}, \texttt{avail(soil, beta)}\} \\
s_{\beta 2} & = & \{\texttt{at(gamma)}, \texttt{avail(soil, beta)}\} \\
s'_{\beta} & = & s_{\beta} \cup \{\texttt{have(soil)}\} \\
s'_{\beta 1} & = & s_{\beta 1} \cup \{\texttt{have(soil)}\} \\
s'_{\beta 2} & = & s_{\beta 2} \cup \{\texttt{have(soil)}\} \\
s_{G\beta 1} & = & s'_{\beta 1} \cup \{\texttt{comm(soil)}\} \\
s_{\gamma} & = & \{\texttt{at(alpha)}, \texttt{avail(soil, gamma)}\} \\
s_{\gamma 1} & = & \{\texttt{at(beta)}, \texttt{avail(soil, gamma)}\} \\
s_{\gamma 2} & = & \{\texttt{at(gamma)}, \texttt{avail(soil, gamma)}\} \\
s'_{\gamma} & = & s_{\gamma} \cup \{\texttt{have(soil)}\} \\
s'_{\gamma 1} & = & s_{\gamma 1} \cup \{\texttt{have(soil)}\} \\
s'_{\gamma 2} & = & s_{\gamma 2} \cup \{\texttt{have(soil)}\} \\
s_{G\gamma 2} & = & s'_{\gamma 2} \cup \{\texttt{comm(soil)}\}
\end{array}$$

Figure 10. States used in the AO* search example.

minimal. The q-value is 3.5 = 1 + (4+1)/2 because the action cost is 1, and the average q-value of the two successors is (4+1)/2 = 2.5. The search generated new nodes, so it must expand the current partial plan in the next iteration.

The search expands the belief states reachable by applying `sample(soil, alpha)` to b_I. Three actions can be applied to the belief state on the left, and the `commun(soil)` action leads to a terminal belief state, whose q-value is 0. Search then generates the successors of the belief state $\{s_{\beta}, s_{\gamma}\}$. Updating the partial solution chooses `commun(soil)` as best for the belief state $\{s'_{\alpha}\}$, and the belief state has its value updated to be 1. Notice that the belief state's value did not change because the heuristic estimate of the value of the belief state was already 1. In general, it is possible for the belief state value to change many times throughout search because the heuristic is an estimate. Then `drive(alpha, beta)` is selected as the best action for the belief state

$\{s_{\beta}, s_{\gamma}\}$ because its q-value is 3, giving the belief state a value of 4 (because the action cost is 1). Then the cost of b_I is updated, but does not change.

Search then expands the current partial solution by first tracing one path to find a terminal belief state $\{s_{G\alpha}\}$. The recursion then traces the plan to find the non-terminal belief state $\{s_{\beta 1}, s_{\gamma 1}\}$ and expands it by generating its successors. The solution is then updated by choosing `sample(soil, beta)` for the newly expanded belief state. The other belief states in the partial plan are also updated, but do not change.

Search again traces the partial plan recursively, and on this iteration generates the successors of the belief states $\{s'_{\beta 1}\}$ and $\{s_{\gamma 1}\}$. Updating chooses `commun(soil)` as the best action for $\{s'_{\beta 1}\}$ (leading to a terminal belief state), and `drive(beta, gamma)` as the best action for $\{s_{\gamma 1}\}$. Over the next two iterations, search generates the successors and selects the best actions for the belief states $\{s_{\gamma 2}\}$ and $\{s'_{\gamma 2}\}$. On the final iteration, search does not generate any new successors because all paths in the current plan lead to terminal belief states. At this point, the subgraph induced by the action selections π is a valid plan, whose average path cost is 3.5.

7. The $POND$ Planner

*AO** serves as the search engine for the $POND$ planner. The $POND$ (partially observable non-deterministic) planner is implemented using several off-the-shelf planning software packages. The components of $POND$ are the PPDDL parser, the IPP planning graph (Koehler *et al.*, 1997), and the CUDD BDD package (Somenzi, 1998) to represent belief states and actions. $POND$ uses modified LAO* (Hansen and Zilberstein, 2001) source code from Eric Hansen to perform AO* search. Even with deterministic actions it is possible to obtain cycles from actions with observations because of planning in belief space. $POND$ constructs the search graph as a directed acyclic graph by employing a cycle-checking algorithm. If adding an edge to the search graph creates a cycle, then the edge cannot represent an action in a strong plan and is hence not added to the graph.

8. Related Work

The following describes related work within classical planning, non-deterministic planning, and probabilistic planning.

8.1. Classical Planning Reachability Heuristics. Given the current popularity of heuristic search planners, it is somewhat surprising to note that the interest in the reachability heuristics in AI planning is a relatively new development. Ghallab and his colleagues were the first to report on a reachability heuristic in IxTeT (Ghallab and Laruelle, 1994) for doing action selection in a partial-order planner. However the effectiveness of their heuristic was not adequately established. Subsequently the idea of reachability heuristics was independently (re)discovered by McDermott (1996, 1999) in the context of his UNPOP planner. UNPOP was one of the first domain-independent planners to synthesize plans containing up to 40 actions. A second independent re-discovery of the idea of using reachability heuristics in planning was made by Bonet and Geffner (1999). Each (re)discovery is the result of attempts to speed up plan synthesis within a different search substrate (partial-order planning, regression, and progression).

The most widely accepted approach to computing reachability information is embodied by GraphPlan. The original GraphPlan planner (Blum and Furst, 1995) used a specialized combinatorial algorithm to search for subgraphs of the planning graph structure that correspond to valid plans. It is interesting to note that almost 75 percent of the original GraphPlan paper was devoted to the specifics of this combinatorial search. Subsequent interpretations of GraphPlan recognized the role of the planning graph in capturing reachability information. Kambhampati *et al.* (1997) explicitly characterized the planning graph as an envelope approximation of the progression search tree (see Figure 2 of his work and the associated discussion). Bonet and Geffner (1999) interpreted GraphPlan as an IDA* search with the heuristic encoded in the planning graph. Nguyen and Kambhampati (2000) described methods for directly extracting heuristics from planning graph. That same year, FF (Hoffmann and Nebel, 2001), a planner using planning graph heuristics, placed first in the International Planning Competition. Since then there has been a steady stream of developments that increased both the effectiveness and the coverage of planning graph heuristics.

8.2. Non-Deterministic Planning. Much of the recent interest in conformant and conditional non-deterministic planning can be traced to CGP (Smith and Weld, 1998), a conformant version of GraphPlan (Blum and Furst, 1995), and SGP (Weld *et al.*, 1998), the analogous conditional version of GraphPlan. Here the graph search is conducted on several planning graphs, each constructed from one of the possible initial states. More recent work on C-plan (Castellini *et al.*, 2001) and Frag-Plan (Kurien *et al.*, 2002) generalize the CGP approach by ordering the searches in the different worlds such that the plan for the hardest to satisfy world is found first, and is then extended to the other worlds. Work described in the next chapter will extend many of the ideas for planning graph based reachability heuristics in classical planning to use planning graphs similar to CGP for reachability heuristics.

Another strand of work models conformant and conditional planning as a search in the space of belief states, as this work pursues. This started with Genesereth and Nourbakhsh (1993), who concentrated on formulating a set of admissible pruning conditions for controlling search. There were no heuristics for choosing among unpruned nodes. GPT (Bonet and Geffner, 2000) extended this search to consider a simple form of reachability heuristic. Specifically, in computing the estimated cost of a belief state, GPT assumes that the initial state is fully observable. The cost estimate itself is done in terms of reachability (with dynamic programming rather than planning graphs). The work described in the next section will approach heuristic search similar to GPT, but with a different heuristic.

Another family of planners that search in belief states is the MBP-family of planners—MBP (Bertoli *et al.*, 2001), and KACMBP (Bertoli and Cimatti, 2002). The MBP-family of planners all represent belief states in terms of binary decision diagrams (BDDs) and action application is modeled as modifications to the BDDs. MBP supports both progression and regression in the space of belief states, while KACMBP is a pure progression planner. This family of planners uses a very simple heuristic, measuring the size of a belief state to indicate the degree of knowledge, and a highly optimized search algorithm to quickly explore the search space.

In contrast to these domain-independent approaches that only require models of the domain physics, PKSPlan (Petrick and Bacchus, 2002) is a forward-chaining *knowledge-based planner* that requires richer domain knowledge. The planner makes use of several knowledge bases, as opposed to a single knowledge base taking the form of a belief state. The knowledge bases separate binary and multi-valued variables, and planning and execution time knowledge.

YAK (Rintanen, 2002) is a regression conditional planner using BDDs that uses a cardinality heuristic. Tuan *et al.* (2004) also present a regression conditional planner that uses an approximation to belief states with 3-valued state variables. While the planner cannot represent all belief states, it is quite efficient for problems where uncertainty can be modeled as unknown propositions.

Non-deterministic planning has also been studied under various extensions, where the unifying theme is guaranteeing plan success in worst-case scenarios. In temporal planning it is especially important to ensure that temporal networks are strongly controllable (Vidal and Ghallab, 1996), similar to strong plans. Musliner *et al.* (1995) explore similar concerns in real-time systems.

8.3. Probabilistic Planning. The earliest work on probabilistic planning focussed on partial order casual link (POCL) planning techniques (Goldman and Boddy, 1994a; Peot and Smith, 1992; Kushmerick *et al.*, 1994; Draper *et al.*, 1994; Onder, 1999). In theory, POCL planners are a nice framework for probabilistic planning because it is easy to add actions to support a low probability condition without backtracking (as may be necessary in state based search). In reality, POCL can be hard to work with because it is often difficult to assess the probability of a partially ordered plan. Due to a lack of scalability, very few recent works has used POCL planners for probabilistic planning (with one notable exception (Onder *et al.*, 2006)). Probabilistic planning has also been studied in the context of GraphPlan (Blum and Langford, 1999), but scalability has only recently improved (Little and Theibaux, 2006).

Most recent work has focussed on MDP and POMDP models (surveyed by Boutilier *et al.* (1999)), where the main interest is in finding algorithms with approximation bounds for optimal solutions to decision theoretic planning. Decision theoretic planning is a generalization of probabilistic planning where rewards are associated with different goal states. Thus, the optimal plan becomes one that can trade-off expected plan cost with expected goal reward. Finding even approximately optimal POMDP solutions is notoriously challenging and the community has moved toward point-based approaches to approximate optimal value functions (Pineau *et al.*, 2003). Solving MDPs has been improved through techniques that intertwine traditional value iteration with heuristic search (Bonet and Geffner, 2003; Barto *et al.*, 1995; Hansen and Zilberstein, 2001; Little *et al.*, 2005; Mausam and Weld, 2005).

Upon recognizing that the probabilistic planning techniques, mentioned above, are extremely computationally costly (Madani *et al.*, 1999), some researchers focussed on restricted classes of plans (Littman, 1997). Several works look at bounded length plan optimization, where the objective is to maximize the probability of goal satisfaction. These include both conformant

(Hyafil and Bacchus, 2004; Majercik and Littman, 1998; Huang, 2006) and conditional (Majercik and Littman, 2003) planners.

The challenge of scaling probabilistic planning still faces all of these approaches. Very few, if any, use the types of reachability heuristics developed in classical planning. Their performance even pales compared to non-deterministic planners in many cases. However, the increased expressivity afforded by probabilistic planning is important. This disconnect between the two classes of planning was the subject of a much heated debate during a 2003 ICAPS workshop on planning under uncertainty that has yet to be resolved sufficiently. While some works have sought out ways to combine the strengths of non-deterministic and probabilistic planners (Mausam *et al.*, 2007), this work develops techniques that are not necessarily specific to either and can benefit both.

CHAPTER 3

PLANNING GRAPH HEURISTICS FOR BELIEF SPACE SEARCH

This chapter presents many of the key intuitions behind reachability heuristics for guiding search in belief state space. By defining the properties of plan distance metrics between belief states, several planning graph heuristic based approaches to estimating the distance metrics are presented. This chapter focusses on non-deterministic conformant and conditional planning with deterministic actions.

1. Introduction

In this chapter I focus on a type of uncertainty, called state incompleteness, where an agent starts in an uncertain (non-deterministic) state but has deterministic actions. The agent seeks strong plans, where they will reach the goal with certainty despite its incomplete state. Ever since CNLP (Peot and Smith, 1992), Plinth (Goldman and Boddy, 1994b), CGP (Smith and Weld, 1998), and SGP (Weld *et al.*, 1998) a series of planners have been developed for tackling such conformant and conditional planning problems – including GPT (Bonet and Geffner, 2000), C-Plan (Castellini *et al.*, 2001), PKSPlan (Petrick and Bacchus, 2002), Frag-Plan (Kurien *et al.*, 2002), MBP (Bertoli *et al.*, 2001), KACMBP (Bertoli and Cimatti, 2002), CFF (Brafman and Hoffmann, 2004), and YAK (Rintanen, 2002). Several of these planners are extensions of heuristic state space planners that search in the space of belief states. Many of the aforementioned planners find strong plans, and heuristic search planners are currently among the best. Yet a foundation for what constitutes a good distance-based heuristic for belief space has not been adequately investigated.

Belief Space Heuristics: Intuitively, it can be argued that the heuristic merit of a belief state depends on at least two factors–the size of the belief state (i.e., the uncertainty in the current state), and the distance of the individual states in the belief state from a goal state. The question of course is how to compute these measures and which are most effective. Many approaches estimate belief state distances in terms of individual state to state distances between states in two belief states, but either lack effective state to state distances or ways to aggregate the state distances. For instance the MBP planner (Bertoli *et al.*, 2001) counts the number of states in the current belief state. This amounts to assuming each state distance has unit cost, and planning for each state can be done independently. The GPT planner (Bonet and Geffner, 2000) measures the state to state distances exactly and takes the maximum distance, assuming the states of the belief state positively interact to achieve the goal.

Heuristic Computation Substrates: I characterize several approaches to estimating belief state distance by describing them in terms of underlying state to state distances. The basis of this investigation is in adapting classical planning reachability heuristics to measure state distances and developing state distance aggregation techniques to measure interaction between plans for the individual states in a belief state. I take two fundamental approaches to measure the distance between a belief state and the goal. The first approach does not involve aggregating state distance measures, rather I use a classical planning graph to compute a representative state distance. The second retains distinctions between individual states in the belief state by using multiple planning graphs, akin to CGP (Smith and Weld, 1998), to compute many state distance measures which are then aggregated. In the next chapter I also describe an improvement upon the multiple planning graph technique in a new planning graph generalization, called the Labeled Uncertainty Graph (LUG), that measures a single distance between a belief state and the goal. This chapter will contain some forward references to the LUG, motivating its development. With these techniques I will discuss the types of heuristics that can be computed, with special emphasis on relaxed plans. The relaxed plan heuristics differ in terms of how they employ state distance aggregation to make stronger assumptions about how states in a belief state can co-achieve the goal through action sequences that are independent, positively interact, or negatively interact.

The motivation for the first of the planning graph techniques for measuring belief state distances is to try a minimal extension to classical planning heuristics to see if they will work. Noticing that using classical planning heuristics ignores distinctions between states in a belief state and may provide uninformed heuristics, I move to the second approach where it is possible to build exponentially many planning graphs to get a better heuristic. With the multiple planning graphs I show how to extract a heuristic from each graph and aggregate them to get the belief state distance measure. By assuming that the states of a belief state are independent, one can aggregate the measures with a summation. Or, by assuming that they positively interact one can use a maximization. However, as I will show, relaxed plans provide a unique opportunity to measure both positive interaction and independence among the states by essentially taking the union of several relaxed plans. Despite the utility of having robust ways to aggregate state distances, we are still faced with the exponential blow up in the number of planning graphs needed. Thus, in the next chapter, I present an approach that seeks to retain the ability to measure the interaction of state distances but avoid computing multiple graphs and extracting heuristics from each. The idea is to condense and symbolically represent multiple planning graphs in a single planning graph, called a *LUG*. Loosely speaking, this single graph unions the causal support information present in the multiple graphs and pushes the disjunction, describing sets of possible worlds (i.e., initial literal layers), into "labels". The planning graph vertices are the same as those present in multiple graphs, but redundant representation is avoided. For instance an action that was present in all of the multiple planning graphs would be present only once in the LUG and labeled to indicate that it is applicable in a planning graph projection from each possible world. We will describe how to extract heuristics from the LUG that make implicit assumptions about state interaction without explicitly aggregating several state distances.

Ideally, each of the planning graph techniques considers every state in a belief state to compute heuristics, but as belief states grow in size this could become uninformed or costly. For example, the single classical planning graph ignores distinctions between possible states where the heuristic based on multiple graphs leads to the construction of a planning graph for each state. One way to keep costs down is to base the heuristics on only a subset of the states in our belief state. We evaluate the effect of such a sampling on the cost of our heuristics. With a single graph we sample a single state and with multiple graphs and the LUG we sample some percent of the states. We evaluate state sampling to show when it is appropriate, and find that it is dependent on how we compute heuristics with the states.

Standardized Evaluation of Heuristics: An issue in evaluating the effectiveness of heuristic techniques is the many architectural differences between planners that use the heuristics. It is quite hard to pinpoint the global effect of the assumptions underlying their heuristics on performance. For example, GPT is outperformed by MBP–but it is questionable as to whether the credit for this efficiency is attributable to the differences in heuristics, or differences in search engines (MBP uses a BDD-based search). The interest in this chapter is to also systematically evaluate a spectrum of approaches for computing heuristics for belief space planning. Thus I have implemented heuristics similar to GPT and MBP and use them to compare against the new heuristics developed around the notion of overlap (multiple world positive interaction and independence). I implemented the heuristics within $POND$. $POND$ does handle search with non-deterministic actions, but in this chapter I discuss deterministic actions. Later chapters will discuss specific approaches needed to handle non-deterministic/probabilistic actions.

The rest of this chapter is organized as follows. I discuss appropriate properties of heuristic measures for belief space planning, and follow with a description of the two planning graph substrates used to compute heuristics. I carry out an empirical evaluation in the next two sections, by describing the test setup, and presenting a standardized internal comparison. I end with related research and various concluding remarks. The next chapter will contain a thorough external evaluation of $POND$ with respect to other state of the art planners, while using the LUG.

2. Reachability Measures in Belief Space

The $POND$ planner uses AO* search to find strong conformant and conditional non-deterministic plans. The AO* algorithm can guide search node expansion with heuristics that estimate the plan distance $dist(b, b')$ between two belief states b and b'. By convention, I assume b is a search node and b' is a belief state comprised of all goal states, called the goal belief state. Since a strong plan (executed in b) ensures that every state $s \in b$ will transition to some state $s' \in b'$, I define the plan distance between b and b' as the number of actions needed to transition every state $s \in b$ to a state $s' \in b$. Naturally, in a strong plan, the actions used to transition a state $s_1 \in b$ may affect the transition of another state $s_2 \in b$. There is usually some degree of positive or negative interaction between s_1 and s_2 that can be ignored or captured in estimating plan distance.[1] In the following I explore how to compute such estimates by using several intuitions from classical planning state distance heuristics.

I start with an example search scenario in Figure 11. There are three belief states b_1 (containing states s_{11} and s_{12}), b_2 (containing state s_{21}), and b_3 (containing states s_{31} and s_{32}). The goal belief state is b_3, and the two progression search nodes are b_1 and b_2. Search should expand the search node with the smallest distance to b_3 by estimating $dist(b_1, b_3)$ – denoted by the bold, dashed line – and $dist(b_2, b_3)$ – denoted by the bold, solid line. Assume for now that the estimates of state distance measures $dist(s, s')$, denoted by the light dashed and solid lines with numbers, are given. The state distances can be referenced by the number of actions or the actual action sequence. In the example, I will use the following action sequences for illustration:

$dist(s_{11}, s_{32}) : (\{a_1, a_2\}, \{a_5\}, \{a_6, a_7\})$,

$dist(s_{12}, s_{31}) : (\{a_1, a_7\}, \{a_3\})$,

[1] Interaction between states captures the notion that actions performed to transition one state to the goal may interfere (negatively interact) or aid with (positively interact) transitioning other states to goals states.

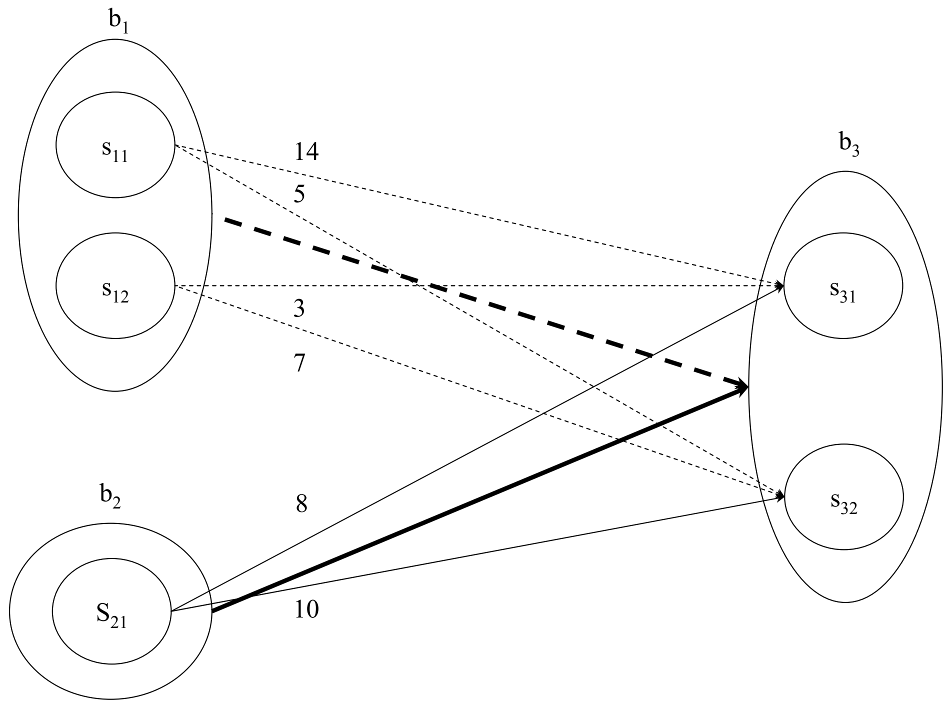


Figure 11. Conformant Plan Distance Estimation in Belief Space

$dist(s_{21}, s_{31}) : (\{a_3, a_6\}, \{a_9, a_2, a_1\}, \{a_0, a_8\}, \{a_5\})$.

In each sequence there may be several actions in each step. For instance, $dist(s_{21}, s_{31})$ has a_3 and a_6 in its first step, and there are a total of eight actions in the sequence – meaning the distance is eight. Notice that the example includes several state distance estimates, which can be found with classical planning techniques. There are many ways to use similar ideas to estimate belief state distance after addressing the issue of belief states containing several states.

Selecting States for Distance Estimation: There exists a considerable body of literature on estimating the plan distance between states in classical planning (Bonet and Geffner, 1999; Nguyen *et al.*, 2002; Hoffmann and Nebel, 2001), and I would like to apply it to estimate the plan distance between two belief states, say b_1 and b_3. I have identified four possible options for using state distance estimates to compute the distance between belief states b_1 and b_3:

- Sample a State Pair: It is possible to sample a single state from b_1 and a single state from b_3, whose plan distance is used for the belief state distance. For example, one might sample s_{12} from b_1 and s_{31} from b_3, then define $dist(b_1, b_3) = dist(s_{12}, s_{31})$.

- Aggregate States: It is possible to form aggregate states for b_1 and b_3 and measure their plan distance. An aggregate state is the union of the propositions in each state in the belief state, defined as:

$$\tilde{s}(b) = \bigcup_{p \in s, s \in b} p$$

  For example, with aggregate states one would compute the belief state distance $dist(b_1, b_3) = dist(\tilde{s}(b_1), \tilde{s}(b_3))$.

- Choose a Subset of States: It is possible to choose a set of states (e.g., by random sampling) from b_1 and a set of states from b_3, and then compute state distances for all pairs of states from the sets. Upon computing all state distances, one can aggregate the state distances (described shortly). For example, one might sample both s_{11} and s_{12} from b_1 and s_{31} from b_3, compute $dist(s_{11}, s_{31})$ and $dist(s_{12}, s_{31})$, and then aggregate the state distances to define $dist(b_1, b_3)$.

- Use All States: One can use all states in b_1 and b_3, and, similar to sampling a subset of states (above), one can compute all distances for state pairs and aggregate the distances.

The former two options for computing belief state distance are reasonably straightforward, given the existing work in classical planning. The latter two options require computing multiple state distances. With multiple state distances there are two details which require consideration in order to obtain a belief state distance measure. In the following I treat belief states as if they contain all of the original states because they can be appropriately replaced with the subset of chosen states.

The first issue is that some of the state distances may not be needed. Since each state in b_1 needs to reach a state in b_3, one should consider the distance for each state in b_1 to "a" state in b_3. However, it may not be necessary to compute the distance for every state in b_1 to reach "every" state in b_3. I will explore assumptions about which state distances need to be computed in Section 2.1.

The second issue, which arises after computing the state distances, is that they must be aggregated into a belief state distance. Popular state distance estimates used in classical planning typically measure aggregate costs of state features (propositions). When searching in belief space, I wish to estimate belief state distance with the aggregate cost of belief state features (states). In Section 2.2, I will examine several choices for aggregating state distances and discuss how each captures different types of state interaction. In Section 2.3, I conclude with a summary of the choices made in order to compute belief state distances.

2.1. State Distance Assumptions. When choosing to compute multiple state distances between two belief states b and b', whether by considering all states or sampling subsets, not all of the state distances are important. For a given state in b it is not necessary to know the distance to every state in b' because each state in b need only transition to one state in b'. There are two assumptions about the states reached in b' which help define two different belief state distance measures in terms of aggregate state distances:

- The first is to optimistically assume that each of the states $s \in b$ can reach the closest of the states $s' \in b'$. With this assumption it is possible to compute distance as:

  $$dist(b, b') = \bigtriangledown_{s \in b} \min_{s' \in b'} dist(s, s').$$

- The second is to assume that all of the states $s \in b$ reach the same state $s' \in b'$, where the aggregate distance is minimum. With this assumption it is possible to compute distance as:

$$dist(b, b') = \min_{s' \in b'} \bigtriangledown_{s \in b} dist(s, s'),$$

where $\bigtriangledown$ represents an aggregation technique (several of which I will discuss shortly).

Throughout the rest of this work I use the first definition for belief state distance because it is relatively robust and easy to compute. Its only drawback is that it treats the states in b in a more independent fashion, but is flexible in allowing states in b to transition to different states in b'. The second definition measures more dependencies of the states in b, but restricts them to reach the same state in b'. While the second may sometimes be more accurate, it is misinformed in cases where all states in b cannot reach the same state s' (i.e., the measure would be infinite). I do not pursue the second method because it may return distance measures that are infinite when they are in fact finite.

In Section 3.3, when I discuss computing these measures with planning graphs, it is possible to implicitly find for each state in b the closest state in b', so that one does not enumerate the states s' in the minimization term of the first belief state distance (above). Part of the reason this is possible is that I compute distance in terms of goal propositions G rather than actual states. Also, because I only consider propositions of b', when I discuss sampling belief states to include in distance computation it is possible to only sample from b. It is also possible to avoid the explicit aggregation $\bigtriangledown$ by using the LUG (described in the next chapter), but I describe several choices for $\bigtriangledown$ here to understand implicit assumptions made by the heuristics computed on the LUG.

2.2. State Distance Aggregation. The aggregation function $\bigtriangledown$ plays an important role in measuring the distance between belief states. When computing more than one state distance measure, either exhaustively or by sampling a subset (as previously mentioned), it is necessary to combine the measures by some means, denoted $\bigtriangledown$. There is a range of options for taking the state distances and aggregating them into a belief state distance. I discuss several assumptions associated with potential measures:

- Positive Interaction of States: Positive interaction assumes that the most difficult state in b requires actions that will help transition all other states in b to some state in b'. In the example, this means that it is assumed that the actions used to transition s_{11} to s_{32} will help us transition s_{12} to s_{31} (assuming each state in b_1 transitions to the closest state in b_3). Inspecting the action sequences, it is easy to see they positively interact because both need actions a_1 and a_7. It is not necessary to know the action sequences to assume positive interaction because it is possible to define the aggregation $\bigtriangledown$ as a maximization of numerical state distances:

  $$dist(b, b') = \max_{s \in b} \min_{s' \in b'} dist(s, s').$$

  The belief state distances are defined as $dist(b_1, b_3) = \max(\min(14, 5), \min(3, 7)) = 5$ and $dist(b_2, b_3) = \max(\min(8, 10)) = 8$. In this case b_1 is preferred over b_2. If each state distance is admissible and one does not sample from belief states, then assuming positive interaction is also admissible.

- Independence of States: Independence assumes that each state in b requires actions that are different from all other states in b in order to reach a state in b'. Previously, there was positive interaction in the action sequences to transition s_{11} to s_{32} and s_{12} to s_{31} because they shared actions a_1 and a_7. There is also some independence in these sequences because the first contains a_2, a_5, and a_6, where the second contains a_3. Again, there is no need to know the action sequences to assume independence because it is possible to define the aggregation $\bigtriangledown$ as a summation of numerical state distances:

  $$dist(b, b') = \sum_{s \in b} \min_{s' \in b'} dist(s, s').$$

  In the example, $dist(b_1, b_3) = \min(14, 5) + \min(3, 7) = 8$, and $dist(b_2, b_3) = \min(8, 10) = 8$. In this case there is no preference over b_1 and b_2.

  Using the cardinality of a belief state $|b| = |\{s|s \in b\}|$ to measure $dist(b, b')$ is a special case of assuming state independence, where $\forall s, s' dist(s, s') = 1$. Using cardinality to measure distance in the example, gives $dist(b_1, b_3) = |b_1| = 2$, and $dist(b_2, b_3) = |b_2| = 1$. With cardinality, b_2 is preferred over b_1.

- Overlap of States: Overlap assumes that there is both positive interaction and independence between the actions used by states in b to reach a state in b'. The intuition is that some actions can often be used for multiple states in b simultaneously and these actions should be counted only once. For example, when computing $dist(b_1, b_3)$ by assuming positive interaction, I noted that the action sequences for $dist(s_{11}, s_{32})$ and $dist(s_{12}, s_{31})$ both used a_1 and a_7. When aggregating these sequences it is best to count a_1 and a_7 each only once because they potentially overlap. However, truly combining the action sequences for maximal overlap is a plan merging problem (Kambhampati *et al.*, 1996), which can be as difficult as planning. Since the ultimate intent is to compute heuristics, it is possible to take a very simple approach to merging action sequences. I introduce a plan merging operator $\Cup$ for $\bigtriangledown$ that picks a step at which to align the sequences and then unions the aligned steps. The number of actions in the resulting action sequence measures belief state distance:

  $$dist(b, b') = \Cup_{s \in b} \min_{s' \in b'} dist(s, s').$$

  To define $\Cup$, I assume that sequences used in progression search start at the same time. Thus, all sequences are aligned at the first step before unioning the steps.

  For example, merging the two plans results in the sequence $dist(s_{11}, s_{32}) \Cup dist(s_{12}, s_{31}) = (\{a_1, a_2\}, \{a_5\}, \{a_6, a_7\}) \Cup (\{a_1, a_7\}, \{a_3\}) = (\{a_1, a_2, a_7\}, \{a_5, a_3\}, \{a_6, a_7\})$ because the sequences are aligned at their first steps, and each step thereafter. Notice that this resulting sequence has seven actions, giving $dist(b_1, b_3) = 7$, whereas defining $\bigtriangledown$ as maximum gave a distance of 5 and as summation gave a distance of 8. Compared with overlap, positive interaction tends to under estimate distance, and independence tends to over estimate distance. As I will show during the empirical evaluation (in Section 5.3), accounting for overlap provides more accurate distance measures for many conformant planning domains.

- Negative Interaction of States: Negative interaction between states can appear in the example if transitioning state s_{11} to state s_{32} makes it more difficult (or even impossible) to transition state s_{12} to state s_{31}. This could happen if performing action a_5 for s_{11} conflicts with action a_3 for s_{12}. It is possible that b_1 cannot reach b_3 if all possible action sequences that start in s_{11} and s_{12}, respectively, and end in any $s \in b_3$ negatively interact.

  There are two ways negative interactions play a role in belief state distances. Negative interactions can prove it is impossible for a belief state b to reach a belief state b', meaning $dist(b, b') = \infty$, or they can potentially increase the distance by a finite amount. In this work, I do not go into the details of computing negative interactions in heuristics and refer the reader to Bryce *et al.* (2006). As such, I do not provide a concrete definition for $\bigtriangledown$ to measure negative interaction.

  While I do not explore ways to adjust the distance measure for negative interactions, I mention some possibilities. Like work in classical planning (Nguyen *et al.*, 2002), it is possible to penalize the distance measure $dist(b_1, b_3)$ to reflect additional cost associated with serializing conflicting actions. Additionally in conditional planning, conflicting actions can be conditioned on observations so that they do not execute in the same plan branch. A distance measure that uses observations would reflect the added cost of obtaining observations, as well as the change in cost associated with introducing plan branches (e.g., measuring average branch cost).

State Selection	State Distance Aggregation	Planning Graph	Heuristic
Single	+ Interaction	SG	Max
Aggregate	Independence	MG	Sum
Subset	Overlap	LUG	Relaxed Plan
All	- Interaction		

Table 3. Features for a belief state distance estimation.

The above techniques for belief state distance estimation in terms of state distances provide the basis for the use of multiple planning graphs. I will show in the empirical evaluation that these measures affect planner performance very differently across standard conformant and conditional planning domains. While it can be quite costly to compute several state distance measures, understanding how to aggregate state distances sets the foundation for techniques I develop in the LUG in the next chapter. As I have already mentioned, the LUG conveniently allows implicit aggregation of state distances to directly measure belief state distance.

2.3. Summary of Methods for Distance Estimation. Since this section explores several methods for computing belief state distances on planning graphs, I provide a summary of the choices to consider, listed in Table 3. Each column is headed with a choice, containing possible options below. The order of the columns reflects the order in which the options are considered. In this section I have covered the first two columns which relate to selecting states from belief states for distance computation, as well as aggregating multiple state distances into a belief state distance. Options for both of these choices are evaluated in the empirical evaluation.

In the next section I will also expand upon how to aggregate distance measures as well as discuss the remaining columns of Table 3. I will present two types of planning graphs in this section: the single planning graph (SG), and multiple planning graphs (MG). In the next chapter I will discuss the labeled uncertainty graph (LUG). Finally, there are many different heuristics to compute on the planning graphs to measure state distances – max, sum, and relaxed plan – which correspond to their classical planning counterparts. I will focus solely on relaxed plans because they are the most effective.

3. Heuristic Measures of Belief State Distances

This section discusses how to use planning graph heuristics to measure the belief state distances defined in the previous section. It covers two types of planning graphs and the extent to which they can be used to compute various heuristics. I begin with a brief background on planning graphs in planning under uncertainty.

Planning Graphs for Belief Space: Planners such as CGP (Smith and Weld, 1998) and SGP (Weld *et al.*, 1998) adapt the GraphPlan idea of compressing the search space with a planning graph by using multiple planning graphs, one for each possible world in the initial belief state. CGP and SGP search on these planning graphs, similar to GraphPlan, to find conformant and conditional plans. This chapter seeks to apply the idea of extracting search heuristics from planning graphs, previously used in state space search (Nguyen *et al.*, 2002; Hoffmann and Nebel, 2001; Bonet and Geffner, 1999) to belief space search.

This section proceeds by describing three classes of heuristics to estimate belief state distance: NG, SG, and MG. NG heuristics are techniques existing in the literature that are not based on planning graphs, SG heuristics are techniques based on a single classical planning graph, MG heuristics are techniques based on multiple planning graphs (similar to those used in CGP). The LUG (described in the next chapter) combines the advantages of SG and MG to reduce the representation size and maintain informedness.

Note that I do not include observations in any of the planning graph structures as would SGP (Weld *et al.*, 1998). Search for conditional plans directly uses the planning graph heuristics by

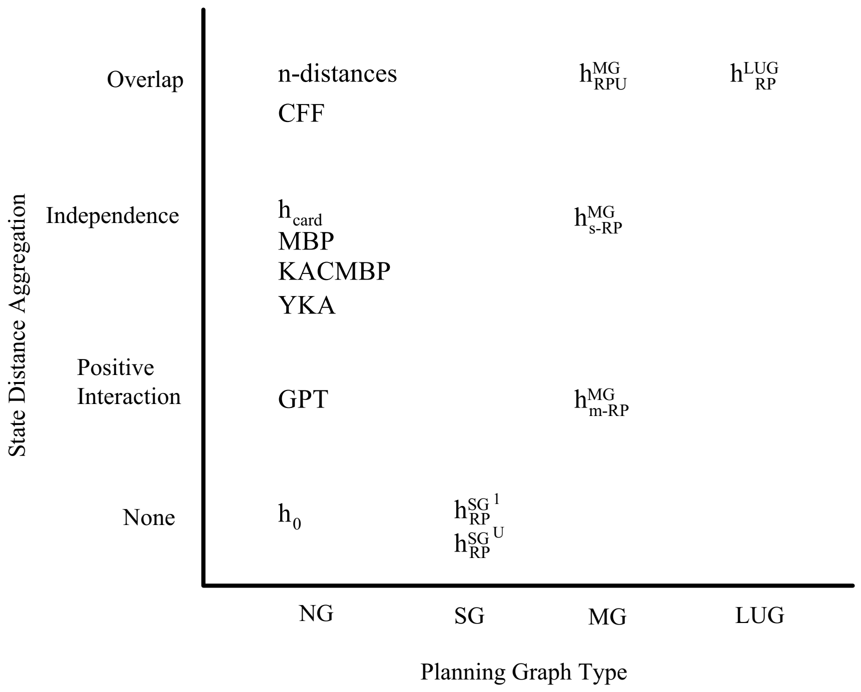


Figure 12. Taxonomy of heuristics with respect to planning graph type and state distance aggregation. Blank entries indicate that the combination is meaningless or not possible.

ignoring observations, and the results show that this still gives good performance. Conformant planning heuristics perform well in conditional planning under the following assumption: a conditional plan and a conformant plan for the same problem will use the same actions, modulo sensory actions. A conditional plan divides the actions among the plan branches, and a conformant plan sequences the actions. If the conformant planning heuristic estimates this set of actions, then in conformant planning the heuristic estimates both the plan length and the number of search node expansions needed to find a plan. In conditional planning, this same heuristic may be a poor estimate of the average branch length, but it will still be a good estimate of the number of search node expansions (where the expansions are divided across the plan branches). While using a conformant planning heuristic for conditional planning may lead to lower quality plans, it may do so with fewer node expansions.

Figure 12 presents a taxonomy of distance measures for belief space. The taxonomy also includes related planners, whose distance measures will be characterized in this section. All of the related planners are listed in the NG group, despite the fact that some actually use planning graphs, because they do not clearly fall into one of my planning graph categories. The figure shows how different substrates (horizontal axis) can be used to compute belief state distance by aggregating state to state distances under various assumptions (vertical axis). Some of the combinations are not considered because they do not make sense or are impossible. The reasons for these omissions will be discussed in subsequent subsections. While there are a wealth of different heuristics one can compute using planning graphs, I concentrate on relaxed plans because they have proven to be the most effective in classical planning and in my previous studies (Bryce and Kambhampati, 2004).

Example: To illustrate the computation of each heuristic, I will use a conformant version of the rover example. Similar to the example used to illustrate search in the previous chapter, each action is deterministic and here there are no observations. The rover is unsure of which location `alpha`, `beta`, or `gamma` will have soil, allowing it to achieve its goal of `comm(soil)`. The initial (non-deterministic) belief state is $b_I = \{s_\alpha, s_\beta, s_\gamma\}$ and the optimal conformant plan to achieve the goal is:

```
 (  sample(soil, alpha),  drive(alpha, beta),
    sample(soil, beta),   drive(beta, gamma),
    sample(soil, gamma),  commun(soil) ),
```

Ideally, the heuristic estimate of the plan cost to reach a belief state where each state satisfies the goal should be six, the same as the optimal plan length.

In the previous section I discussed the important issue of sampling states to include in the heuristic computation. I will not directly address this issues in this section, deferring it to discussion in the empirical evaluation section. The heuristics below are computed after deciding on a set of states to use, whether by sampling or not. Also, as previously mentioned, I only consider sampling states from the belief state generated by search because it is possible to implicitly find closest states from the goal belief state without sampling.

I proceed by describing the various substrates used for computing belief space distance estimates. Within each I describe the prospects for various types of possible world aggregation. In addition to the heuristics developed in this work, I mention related work where relevant.

3.1. Non Planning Graph-based Heuristics (NG).

I group many heuristics and planners into the NG group because they are not using SG, MG, or LUG planning graphs. Just because they are mentioned in this group it does not mean they are not using planning graphs in some other form.

No Aggregation: Breadth first search uses a simple heuristic, h_0 where the heuristic value is set to zero. I mention this heuristic to gauge the effectiveness of the search algorithm relative to improvements gained through using heuristics.

Positive Interaction Aggregation: The GPT planner (Bonet and Geffner, 2000) measures belief state distance as the maximum of the minimum state to state distance of states in the source and destination belief states, assuming optimistic reachability as mentioned in Section 2.2. GPT measures state distances exactly, in terms of the minimum number of transitions in the state space. Taking the maximum state to state distance is akin to assuming positive interaction of states in the current belief state.

Independence Aggregation: The MBP planner (Bertoli *et al.*, 2001), KACMBP planner (Bertoli and Cimatti, 2002), YAK planner (Rintanen, 2002), and my definition of a comparable h_{card} heuristic measure belief state distance by assuming every state to state distance is one, and taking the summation of the state distances (i.e., counting the number of states in a belief state). Cardinality is useful in progression because as belief states become smaller, the agent has more knowledge and it can be easier to reach a goal state.

In the conformant rover problem, $h_{card}(b_I, G) = 3$ because b_I has three states consistent with it. Notice, this may be uninformed because every reachable belief state will have three states, as it is impossible to ever know which location has a soil sample. In general, cardinality does not accurately reflect distance measures. For instance, h_{card} reverts to uninformed search in classical planning problems because state distance may be large or small but it still assigns a heuristic value of one to every state.

Overlap Aggregation: Rintanen (2004) describes n-Distances which generalize the belief state distance measure in GPT to consider the maximum n-tuple state distance. The measure involves, for each n-sized tuple of states in a belief state, finding the length of the actual plan to transition the n-tuple to a belief state satisfying the goal. Then the maximum n-tuple distance is taken as the distance measure.

For example, consider a belief state with four states. With an n equal to two, it is possible to define six belief states, one for each size two subset of the four states. A real plan is found for each of these belief states, and the maximum cost of these plans is used to measure the distance for the original four state belief state. When n is one, the heuristic is equivalent to the measure used by GPT, and when n is equal to the size of the belief state it is equivalent to directly solving the planning problem. While it is costly to compute this measure for large values of n, it is very informed as it accounts for overlap and negative interactions.

The CFF planner (Brafman and Hoffmann, 2004) uses a version of a relaxed planning graph to extract relaxed plans. The relaxed plans measure the cost of supporting the goal from all states in a belief state. In addition to the traditional notion of a relaxed planning graph that ignores negative interactions, CFF also ignores all but one antecedent proposition in conditional effects to keep its

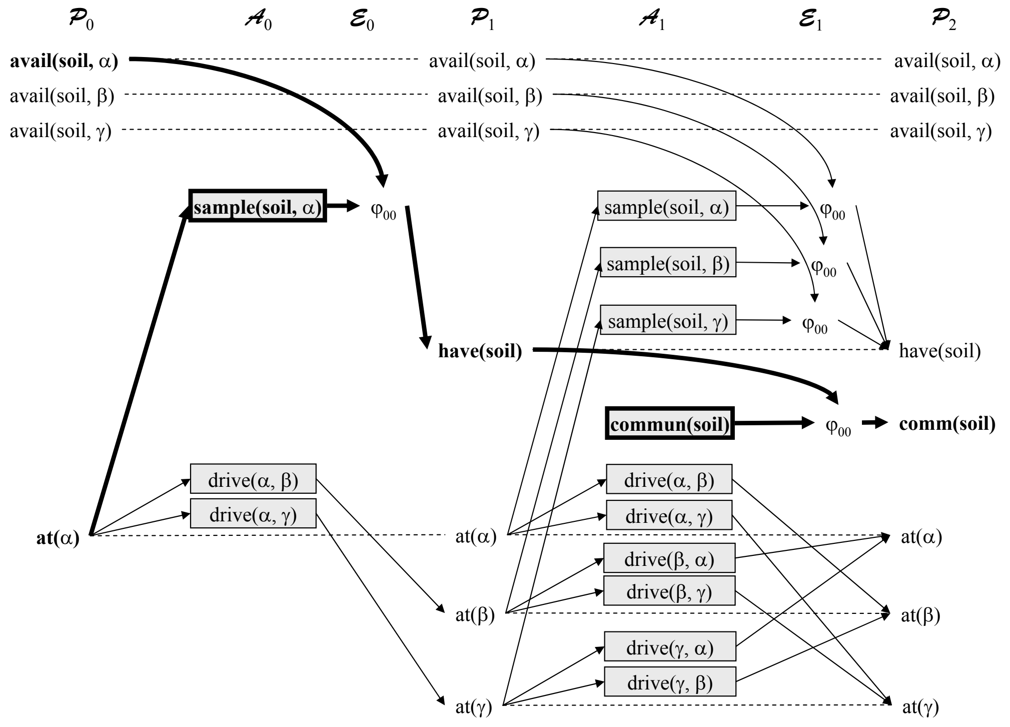


Figure 13. Single planning graph for conformant rovers, with relaxed plan components in bold.

relaxed plan reasoning tractable. The CFF relaxed plan does capture overlap but ignores some subgoals and all negative interactions. The way CFF ensures the goal is supported in the relaxed problem is to encode the relaxed planning graph as a satisfiability problem. If the encoding is satisfiable, the chosen number of action assignments is the distance measure.

3.2. Single Planning Graph Heuristics (**SG**). The simplest approach for using planning graphs for belief space planning heuristics is to use a "classical" planning graph. To form the initial proposition layer from a belief state, it is possible to either sample a single state (denoted SG^1) or use an aggregate state (denoted SG^U).

When computing the heuristic for b_I in the conformant rovers problem (see Figure 13), it possible to define $\mathcal{P}_0$ for SG^1 by sampling a state (say s_α):

$\mathcal{P}_0$ = {`at(alpha), avail(soil, alpha)`},

or define $\mathcal{P}_0$ for SG^U such that it is equal to $\hat{s}(b_I)$

$\mathcal{P}_0$ = {`at(alpha), avail(soil, alpha),`

`avail(soil, beta), avail(soil, gamma)`}

Since these two versions of the single planning graph have identical semantics, aside from the initial literal layer, I proceed by describing the SG^U graph and point out differences with SG^1 where they arise.

Graph construction is similar to classical planning graphs and stops when all goal propositions appear in a proposition layer. Unlike, the previous chapter, I use the planning graph formalism used in IPP (Koehler *et al.*, 1997) to allow for explicit representation of conditional effects, meaning there is a proposition layer $\mathcal{P}_k$, an action layer $\mathcal{A}_k$, and an effect layer $\mathcal{E}_k$ in each level k of a planning graph. The planning graph is a sequence of layers ($\mathcal{A}_0$, $\mathcal{E}_0$, $\mathcal{P}_1$, ..., $\mathcal{A}_k$, $\mathcal{E}_k$, $\mathcal{P}_{k+1}$). Persistence for a proposition p, denoted by a_p, is represented as an action where $\rho_e(a_p) = \varepsilon_{00}(a_p) = p$. A proposition is in $\mathcal{P}_k$ if an effect from the previous effect layer $\mathcal{E}_{k-1}$ contains the proposition in its consequent. An action is in the action layer $\mathcal{A}_k$ if every one of its execution precondition propositions is in $\mathcal{P}_k$. An effect is in the effect layer $\mathcal{E}_k$ if its associated action is in the action layer $\mathcal{A}_k$ and every one of its antecedent propositions is in $\mathcal{P}_k$. Using conditional effects in the planning graph avoids factoring an action with conditional effects into a possibly exponential number of non-conditional actions, but adds an extra planning graph layer per level. Note that, to simplify the illustration, Figure 13 (and later planning graph figures) only depicts conditional effects in the effect layer (e.g., φ_{00}(`sample(soil, alpha)`)) and leaves unconditional effects (e.g., φ_{00}(`drive(alpha, beta)`)) implicit. Once the graph is built, one can extract heuristics.

No Aggregation: Relaxed plans within a single planning graph are able to measure, under the most optimistic assumptions, the distance between two belief states. The relaxed plan represents a distance between a subset of the initial layer propositions and the propositions in the goal. In the SG^U, the propositions from the initial layer that are used for support may not hold in any single state of the projected belief state, unlike the SG^1. The classical relaxed plan heuristic h^{SG}_{RP} finds a set of (possibly interfering) actions to support the goal propositions. The relaxed plan RP is a subgraph of the planning graph, of the form ($\mathcal{A}^{RP}_0$, $\mathcal{E}^{RP}_0$, $\mathcal{P}^{RP}_1$, ..., $\mathcal{A}^{RP}_k$, $\mathcal{E}^{RP}_k$, $\mathcal{P}^{RP}_{k+1}$). Each of the layers contains a subset of the vertices in the corresponding layer of the planning graph.

The only change in the relaxed plan extraction, over the algorithm described in the previous chapter, is that it requires reasoning about the conditional effects explicitly. As before, the goal propositions are supported starting from the last layer of the planning graph. Each proposition must be supported by an effect from the previous layer. Each effect must be supported by its associated action. Then, each proposition appearing in a chosen action's precondition, or a chosen effect's antecedent, must be supported. The relaxed plan heuristic value is still the summation of the number of (non-persistence) actions in each action layer of a planning graph with $k + 1$ levels:

$$h^{SG}_{RP}(b, G) = \sum_{j=0}^{k} \mid \mathcal{A}^{RP}_j \mid$$

In the rover problem, the relaxed plan extracted from the SG^U is shown in Figure 13 as the bold edges and nodes. The goal proposition `have(soil)` is supported by the effect of `commun(soil)` in $\mathcal{E}_1$, and the effect is supported by the `commun(soil)` action. In $\mathcal{P}_1$, `have(soil)` is supported (because it is in the antecedent of the effect of `commun(soil)`) by the effect of `sample(soil, alpha)` in $\mathcal{E}_0$. This requires inclusion of the `sample(soil, alpha)` action in $\mathcal{A}_0$ and `avail(soil, alpha)` and `at(alpha)` in $\mathcal{P}_0$. The relaxed plan contains two actions, defining $h^{SG^U}_{RP}(b_I, G)$ = 2. This greatly underestimates the optimal plan length of six because it ignores all but one state of b_I. Sampling a single state, as in SG^1, would lead to a similar result. A single, unmodified classical planning graph cannot capture support from all possible worlds – hence there is no explicit aggregation over distance measures for states. As a result, I do not mention aggregating states to measure positive interaction, independence, or overlap.

3.3. Multiple Planning Graph Heuristics (MG). Single graph heuristics are usually uninformed because the belief state used for constructing the planning graph often corresponds to multiple possible states. The lack of accuracy is because single graphs are not able to capture propagation of multiple world support information. Recall the single graph SG^U for b_I of the conformant rover problem. If `sample(soil, alpha)` and `commun(soil)` were the only actions, one would say that `comm(soil)` is reachable with a cost of two, but in fact the cost is infinite (since there is no `sample(soil, beta)` and `sample(soil, gamma)` to support `have(soil)` from all possible worlds), and there is no strong plan.

To account for lack of support in all possible worlds and sharpen the heuristic estimate, a set of multiple planning graphs is considered. Each is a single graph, as previously discussed. These multiple planning graphs are similar to the planning graphs used by CGP (Smith and Weld, 1998), but lack mutexes. Each planning graph $SG(s) \in MG(b)$ constructs the initial layer $\mathcal{P}_0(s)$ of with a different state $s \in b$. With multiple planning graphs, the heuristic value of a belief state is computed in terms of all the planning graphs. Unlike single planning graphs, it is possible to compute different world aggregation measures with the multiple planning graphs.

While it is possible to get a more informed heuristic by considering more of the states in a belief state, in certain cases it can be costly to compute the full set of planning graphs and extract relaxed plans. I will describe computing the full set of planning graphs, but will later evaluate the effect of computing a smaller proportion of these. The single graph SG^1 is the extreme case of computing a single one of the multiple planning graphs.

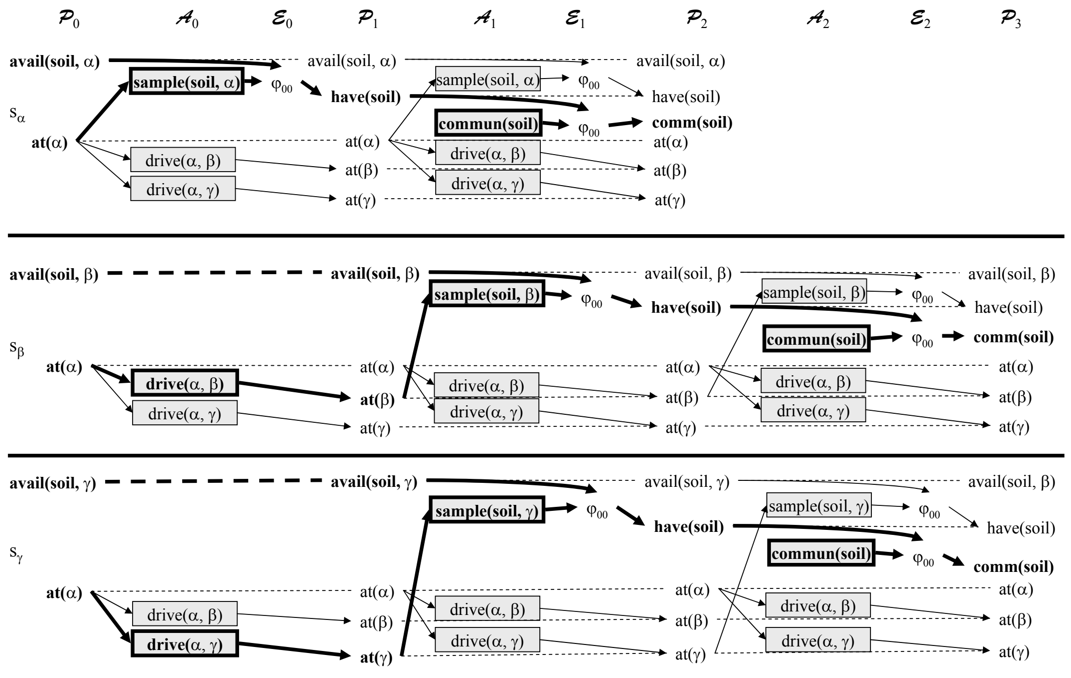


Figure 14. Multiple planning graphs for conformant rovers, with relaxed plan components bolded.

To illustrate the use of multiple planning graphs, consider the conformant rover example. It is possible to build three planning graphs for b_I, one for each state. The respective initial proposition layers are:

$\mathcal{P}_0(s_\alpha) = \{$`at(alpha), avail(soil, alpha)`$\}$,

$\mathcal{P}_0(s_\beta) = \{$`at(alpha), avail(soil, beta)`$\}$, and

$\mathcal{P}_0(s_\gamma) = \{$`at(alpha), avail(soil, gamma)`$\}$.

A single planning graph is sufficient if there is no need to aggregate state measures, so in the following I consider how to compute the goal achievement cost with multiple graphs by aggregating state distances.

Positive Interaction Aggregation: Similar to GPT (Bonet and Geffner, 2000), one can use the worst-case world to represent the cost of the belief state by using the h^{MG}_{m-RP} heuristic. The difference with GPT is that the heuristic is computed on planning graphs, instead of computing plans in state space. This heuristic accounts for the number of actions used in a given world, but assume positive interaction across all possible worlds.

The h^{MG}_{m-RP} heuristic is computed by finding a relaxed plan $RP(s)$ on each planning graph, exactly as done on the single graph with h^{SG}_{RP}. The difference is that unlike the single planning graph SG^U, but like SG^1, the initial levels of the planning graphs are states, so each relaxed plan will reflect all the support needed in the possible world corresponding to a state $s \in b$. Formally:

$$h^{MG}_{m-RP}(b, G) = \max_{s \in b} \left(\sum_{j=0}^{k} \mid \mathcal{A}_j^{RP(s)} \mid \right)$$

where k is the level of $SG(s)$ where the goal propositions are first reachable.

Notice that this heuristic is not computing all state distances between states in b and states consistent with G. Each planning graph $SG(s)$ corresponds to a state in $s \in b$, and the heuristic extracts a single relaxed plan from each. It is not necessary to enumerate all states consistent with the goal and find a relaxed plan for each. The heuristic supports only the propositions in G. These propositions are the estimated minimum distance goal state.

For the conformant rover problem, computing $h^{MG}_{m-RP}(b_I, G)$ (Figure 14) finds the following relaxed plans:

$\mathcal{A}_0^{RP(s_\alpha)} = \{$`sample(soil, alpha)`$\}$,

$\mathcal{E}_0^{RP(s_\alpha)} = \{\varphi_{00}($`sample(soil, alpha)`$)\}$,

$\mathcal{P}_1^{RP(s_\alpha)} = \{$`have(soil)`$\}$,

$\mathcal{A}_1^{RP(s_\alpha)} = \{$`commun(soil)`$\}$,

$\mathcal{E}_1^{RP(s_\alpha)} = \{\varphi_{00}($`commun(soil)`$)\}$,

$\mathcal{P}_2^{RP(s_\alpha)} = \{$`comm(soil)`$\}$

from the second planning graph it finds:

$\mathcal{A}_0^{RP(s_\beta)} = \{\texttt{drive(alpha, beta)}, \texttt{avail(soil, beta)}_p\}$,

$\mathcal{E}_0^{RP(s_\beta)} = \{\varphi_{00}(\texttt{drive(alpha, beta)}), \varphi_{00}(\texttt{avail(soil, beta)}_p)\}$,

$\mathcal{P}_1^{RP(s_\beta)} = \{\texttt{at(beta), avail(soil, beta)}\}$,

$\mathcal{A}_1^{RP(s_\beta)} = \{\texttt{sample(soil, alpha)}\}$,

$\mathcal{E}_1^{RP(s_\beta)} = \{\varphi_{00}(\texttt{sample(soil, beta)})\}$,

$\mathcal{P}_2^{RP(s_\beta)} = \{\texttt{have(soil)}\}$,

$\mathcal{A}_2^{RP(s_\beta)} = \{\texttt{commun(soil)}\}$,

$\mathcal{E}_2^{RP(s_\beta)} = \{\varphi_{00}(\texttt{commun(soil)})\}$,

$\mathcal{P}_3^{RP(s_\beta)} = \{\texttt{comm(soil)}\}$

and from the last planning graph it finds:

$\mathcal{A}_0^{RP(s_\gamma)} = \{\texttt{drive(alpha, gamma), avail(soil, gamma)}_p\}$,

$\mathcal{E}_0^{RP(s_\gamma)} = \{\varphi_{00}(\texttt{drive(alpha, gamma)}), \varphi_{00}(\texttt{avail(soil, gamma)}_p)\}$,

$\mathcal{P}_1^{RP(s_\gamma)} = \{\texttt{at(gamma), avail(soil, gamma)}\}$,

$\mathcal{A}_1^{RP(s_\gamma)} = \{\texttt{sample(soil, alpha)}\}$,

$\mathcal{E}_1^{RP(s_\gamma)} = \{\varphi_{00}(\texttt{sample(soil, gamma)})\}$,

$\mathcal{P}_2^{RP(s_\gamma)} = \{\texttt{have(soil)}\}$,

$\mathcal{A}_2^{RP(s_\gamma)} = \{\texttt{commun(soil)}\}$,

$\mathcal{E}_2^{RP(s_\gamma)} = \{\varphi_{00}(\texttt{commun(soil)})\}$,

$\mathcal{P}_3^{RP(s_\gamma)} = \{\texttt{comm(soil)}\}$

The first relaxed plan contains two actions and the second and third each have three actions. Taking the maximum of the number of actions in the relaxed plan values gives $h^{MG}_{m-RP}(b_I, G) = 3$. This aggregation ignores the fact that the plan must use `sample` actions at different locations.

Independence Aggregation: Using the h^{MG}_{s-RP} heuristic assumes independence among the worlds in the belief state. Relaxed plan extraction is exactly as described in the previous heuristic and the heuristic uses a summation rather than maximization of the relaxed plan costs. Formally:

$$h^{MG}_{s-RP}(b, G) = \sum_{s \in b} \left(\sum_{j=0}^{k} \mid \mathcal{A}^{RP(s)}_j \mid \right)$$

where k is the level of each $SG(s)$ where the goal propositions are first reachable.

For the conformant rovers example, computing $h^{MG}_{s-RP}(b_I, G)$ finds the same relaxed plans as in the $h^{MG}_{m-RP}(b_I, G)$ heuristic, but sums their values to get 2 + 3 + 3 = 8 as the heuristic. This aggregation ignores the fact that the plan can use the same `commun(soil)` action for all possible worlds.

State Overlap Aggregation: Notice that the two previous heuristics either take a maximization and do not account for some actions, or take a summation and possibly account for extra actions. The h^{MG}_{RPU} heuristic can balance the measure between positive interaction and independence of worlds. Examining the relaxed plans computed by the two previous heuristics for the conformant rovers example, one can see that the relaxed plans extracted from each graph have some overlap. Notice, that $\mathcal{A}^{RP(s_\alpha)}_1$, $\mathcal{A}^{RP(s_\beta)}_2$ and $\mathcal{A}^{RP(s_\gamma)}_2$ contain a `commun(soil)` action irrespective of which location has soil – showing some positive interaction. Also, $\mathcal{A}^{RP(s_\alpha)}_0$ contains `sample(soil, alpha)`, $\mathcal{A}^{RP(s_\beta)}_1$ contains `sample(soil, beta)`, and $\mathcal{A}^{RP(s_\gamma)}_1$ contains `sample(soil, gamma)` – showing some independence. Taking the layer-wise union of the three relaxed plans leads to a unioned relaxed plan:

$$\begin{aligned}\mathcal{A}^{RP_U}_0 = \; & \{\texttt{sample(soil, alpha), drive(alpha, beta),} \\ & \texttt{drive(alpha, gamma), avail(soil, beta)}_p\texttt{,} \\ & \texttt{avail(soil, gamma)}_p\}\end{aligned}$$

$$\mathcal{E}_0^{RP_U} = \{\varphi_{00}(\texttt{sample(soil, alpha)}), \varphi_{00}(\texttt{drive(alpha, beta)}),$$
$$\varphi_{00}(\texttt{drive(alpha, gamma)}), \varphi_{00}(\texttt{avail(soil, gamma)}_p)\}$$

$$\mathcal{P}_1^{RP_U} = \{\texttt{have(soil)}, \texttt{at(beta)}, \texttt{at(gamma)},$$
$$\texttt{avail(soil, beta)}, \texttt{avail(soil, gamma)}\}$$

$$\mathcal{A}_1^{RP_U} = \{\texttt{commun(soil)}, \texttt{sample(soil, beta)}, \texttt{sample(soil, gamma)}\}$$

$$\mathcal{E}_1^{RP_U} = \{\varphi_{00}(\texttt{commun(soil)}), \varphi_{00}(\texttt{sample(soil, beta)}),$$
$$\varphi_{00}(\texttt{sample(soil, gamma)})\}$$

$$\mathcal{P}_2^{RP_U} = \{\texttt{comm(soil)}, \texttt{have(soil)}\}$$

$$\mathcal{A}_2^{RP_U} = \{\texttt{commun(soil)}\}$$

$$\mathcal{E}_2^{RP_U} = \{\varphi_{00}(\texttt{commun(soil)})\}$$

$$\mathcal{P}_3^{RP_U} = \{\texttt{comm(soil)}\}$$

This relaxed plan accounts for many of the actions that are the same between possible worlds and the actions that differ. Notice that `commun(soil)` appears only once in level two and the `sample` actions appear in layers zero and one. However, `commun(soil)` also appears in level one because a poor alignment of the possible worlds was chosen.

In order to take the union of relaxed plans, the heuristic extracts relaxed plans from each planning graph, as in the two previous heuristics. The relaxed plans are *start-aligned* to reflect that they all start at the same time. The summation of the number of actions of each action level in the unioned relaxed plan is used as the heuristic value. Formally:

$$h_{RPU}^{MG}(b, G) = \sum_{j=0}^{k} \mid \mathcal{A}_j^{RP_U} \mid$$

where k is the greatest level where the goal propositions are first reachable among all planning graphs.

Problem	Initial States	Goal Propositions	Propositions	Causative Actions	Observational Actions	Optimal Parallel	Optimal Serial
Rovers1	1	1	66	88	0 {12}	5 {5}	5 {5}
Rovers2	2	1	66	88	0 {12}	8 {7}	8 {7}
Rovers3	3	1	66	88	0 {12}	10 {?}	10 {8}
Rovers4	4	1	66	88	0 {12}	13 {?}	13 {10}
Rovers5	16	3	71	97	0 {12}	? {?}	20 {?}
Rovers6	12	3	119	217	0 {18}	? {?}	? {?}
Logistics1	2	1	29	70	0 {10}	6 {6}	9 {7}
Logistics2	4	2	36	106	0 {20}	6 {?}	15 {12}
Logistics3	2	1	58	282	0 {21}	8 {?}	11 {8}
Logistics4	4	2	68	396	0 {42}	8 {?}	18 {?}
Logistics5	8	3	78	510	0 {63}	? {?}	28 {?}
BT(n)	n	1	n+1	n	0 {n}	1 {1}	n {n-1}
BTC(n)	n	1	n+2	n+1	0 {n}	2n-1 {2}	2n-1 {n-1}
CubeCenter(n)	n^3	3	3n	6	0	(3n-3)/2	(9n-3)/2
Ring(n)	$n3^n$	n	4n	4	0	3n-1	3n-1

Table 4. Features of test domains and problems - Number of initial states, Number of goal propositions, Number of propositions, Number of causative actions, Number of Observational Actions, Optimal number of parallel plan steps, Optimal number of serial plan steps. Data for conditional versions of domains is in braces; plan lengths for conditional plans are maximum conditional branch length.

For the conformant rovers problem, the number of actions in RP^U provides a heuristic value of $h^{MG}_{RPU}(b_I, G) = 7$. Notice that this estimate is in between the positive interaction and independence aggregation heuristics.

4. Empirical Evaluation: Domains

This section presents the domains used in the experiments discussed in this chapter and the next chapter. All tests were run in Linux on an x86 machine with a 2.66GHz P4 processor and 1GB RAM with a timeout of 20 minutes. $POND$ uses a heuristic weight of five in the AO* search.

Table 4 shows some of the relative features of the different problems used to evaluate the heuristics. The table shows the number of initial states, goal propositions, propositions, actions, and optimal plan lengths. Some of the propositions are static and are counted in the table. This can be used as a guide to gauge the difficulty of the problems, as well as our performance.

Conformant Problems: In addition to the standard domains used in conformant planning–such as Bomb-in-the-Toilet, Ring, and Cube Center, I also developed two new domains Logistics and Rovers. I chose these new domains because they have more difficult subgoals, and have many plans of varying length.

The Ring domain involves a circular ring of n rooms where each room is connected to two adjacent rooms. Each room has a window which can be open, closed, or locked. The goal is to have every window locked. Initially, any state is possible – the agent could be in any room and each window could be in any configuration. There are four actions: move right, move left, close the window in the current room, and lock the window in the current room. Closing a window only works if the window is open, and locking a window only works if the window is closed. A good conformant plan involves moving in one direction closing and locking the window in each room.

The Cube Center domain involves a three-dimensional grid (cube) where there are six actions – it is possible to move in two directions along each dimension. Each dimension consists of n possible locations. Moving in a direction along which there are no more grid points leaves one in the same position. Using this phenomena, it is possible to localize in each dimension by repeatedly moving in the same direction. Initially it is possible to be at any location in the cube and the goal is to reach the center. A good conformant plan involves localizing in a corner and then moving to the center.

The Rovers domain is a conformant adaptation of the analogous domain of the classical planning track of the International Planning Competition (Long and Fox, 2003). The added uncertainty to the initial state uses conditions that determine whether an image objective is visible from various vantage points due to weather, and the availability of rock and soil samples. The goal is to upload an image of an objective and some rock and soil sample data. Thus a conformant plan requires visiting all of the possible vantage points and taking a picture, plus visiting all possible locations of soil and rock samples to draw samples. The first five Rovers problems have 4 waypoints. Problems one through four have one through four locations, respectively, at which a desired imaging objective is possibly visible (at least one will work, but it is unknown which one). Problem 5 adds some rock and soil samples as part of the goal and several waypoints where one of each can be obtained (again, it is unknown which waypoint will have the right sample). Problem 6 adds two more waypoints, keeps the same goals as Problem 5 and changes the possible locations of the rock and soil samples. In all cases the waypoints are connected in a tree structure, as opposed to completely connected.

The Logistics domain is a conformant adaptation of the classical Logistics domain where trucks and airplanes move packages. The uncertainty is the initial locations of packages. Thus, any actions relating to the movement of packages have a conditional effect that is predicated on the package actually being at a location. In the conformant version, the drivers and pilots cannot sense or communicate a package's actual whereabouts. The problems scale by adding packages and cities. The Logistics problems consist of one airplane, and cities with an airport, a post office, and a truck. The airplane can travel between airports and the trucks can travel within cities. The first problem has two cities and one package that could start at either post office, and the goal is to get the package to the second city's airport. The second problem adds another package at the same possible starting points and having the same destination. The third problem has three cities with one package that could be at any post office and has to reach the third airport. The fourth problem adds a second package to the third problem with the same starting and ending locations. The fifth problem has three cities and three packages, each at one of two of the three post offices and having to reach different airports.

Conditional Problems: For conditional planning I consider domains from the literature: Bomb-in-the-Toilet with sensing (BTS), and Bomb-in-the-Toilet with clogging and sensing (BTCS). I also extend the conformant Logistics and Rovers to include sensory actions.

The Rovers problem allows for the rover, when it is at a particular waypoint, to sense the availability of image, soil, or rock data at that location. The locations of the collectable data are expressed as one-of constraints, so the rover can deduce the locations of collectable data by failing to sense the other possibilities. For example by knowing that one of `avail(soil, alpha)`, `avail(soil, beta)`, or `avail(soil, gamma)` is true, then by sensing both `avail(soil, alpha)` and `avail(soil, beta)` as false we can deduce `avail(soil, gamma)` is true.

Logistics has observations to determine if a package at a location exists, and the observation is assumed to be made by a driver or pilot at the particular location. Since there are several drivers and a pilot, different agents make the observations. The information gained by the agents is assumed to be automatically communicated to the others, as the planner is the agent that has all the knowledge.[2]

5. Empirical Evaluation: Inter-Heuristic Comparison

In this section I intend to evaluate which distance estimates perform best, both in terms of overall planner performance and accuracy. I compare many techniques within $POND$ by using the conformant and conditional domains. The performance metrics include the total planning time and the number of search nodes expanded. I proceed by presenting how $POND$ performs with the different heuristics in both conformant and conditional domains, explore the effect of sampling a proportion of worlds to build SG^1, MG, and LUG graphs, and compare the heuristic estimates in $POND$ to the optimal plan length to gauge heuristic accuracy. I finish with a summary of important conclusions. I mention some results for the LUG in this section, while it is presented in the next

[2]This problem may be interesting to investigate in a multi-agent planning scenario, assuming no global communication (e.g. no radio dispatcher).

chapter, because they identify how an improved representation of multiple planning graphs can reduce computational cost while still computing the same heuristics.

With the exception of the discussion on sampling worlds to construct the planning graphs, the planning graphs are constructed deterministically. This means that the single graph is the unioned single graph SG^U, and the MG and LUG graphs are built for all possible worlds.

5.1. Heuristic Measure Comparison. In Tables 5 and 6 and Figures 15 to 23, I show results in terms of the total time and number of expanded nodes. The total time captures the amount of time computing the heuristic and searching. A high total time with a high number of search nodes indicates a poor heuristic, and a high total time and low number of search nodes indicates an expensive but informed heuristic. I describe the results from left to right in Table 5, comparing the different planning graph structures and relaxed plans computed on each planning graph.

I start with the non-planning graph heuristics h_0 and h_{card}. As expected, h_0, breadth-first search, does not perform well in a large portion of the problems, shown by the large number of search nodes and inability to scale to solve larger problems. In the BT and BTC domains (Figures 17 and 18) the h_{card} heuristic does not work well because the size of belief states does not change as the search gets closer to the goal (it is impossible to ever know which package contains the bomb). Also, h_{card} does not perform as well in the Rovers and Logistics problems (Figures 15 and 16) because the size of a belief state, during planning, does not necessarily indicate that the belief state will be in a good plan. Part of the reason h_{card} works so well in some domains like Ring and Cube (Figures 20 and 19) is that it measures knowledge, and plans for these domains are largely based on increasing knowledge. The reason h_{card} performs poorly on other domains is that finding causal support (which it does not measure) is more important than knowledge for these domains.

With a single planning graph SG^U, $POND$ does reasonably well with the h^{SG}_{RP} heuristic

Problem	h_0	h_{card}	h_{RP}^{SG}	h_{m-RP}^{MG}	h_{s-RP}^{MG}	h_{RPU}^{MG}
Rovers 1	540/36	520/21	590/6	580/6	580/6	580/6
2	940/249	790/157	700/15	1250/32	750/10	830/13
3	3340/1150	2340/755	3150/230	3430/77	1450/24	1370/23
4	TO	14830/4067	13480/1004	10630/181	7000/163	2170/34
5	-	TO	TO	85370/452	12470/99	31480/73
6	-	-	-	180890/416	15780/38	31950/73
Logistics 1	560/169	530/102	680/46	970/58	730/21	650/9
2	TO	TO	TO	2520/32	6420/105	2310/20
3	-	-	-	27820/927	4050/83	2000/15
4	-	-	-	5740/27	29180/211	53470/382
5	-	-	-	42980/59	51380/152	471850/988
BT 2	450/3	460/2	460/3	450/2	450/2	500/2
10	760/1023	590/428	1560/1023	6200/428	820/10	880/10
20	TO	TO	TO	TO	6740/20	6870/20
30	-	-	-	-	41320/30	44260/30
40	-	-	-	-	179930/40	183930/40
50	-	-	-	-	726930/50	758140/50
60	-	-	-	-	TO	TO
70	-	-	-	-	-	-
80	-	-	-	-	-	-
BTC 2	460/5	460/4	450/5	460/4	460/3	470/3
10	1090/2045	970/1806	3160/2045	18250/1806	980/19	990/19
20	TO	TO	TO	TO	TO	9180/39
30	-	-	-	-	-	54140/59
40	-	-	-	-	-	251140/79
50	-	-	-	-	-	1075250/99
60	-	-	-	-	-	TO
70	-	-	-	-	-	-
CubeCenter 3	10/184	30/14	90/34	1050/61	370/9	0430/11
5	180/3198	20/58	3510/1342	60460/382	11060/55	14780/82
7	1940/21703	40/203	46620/10316	TO	852630/359	1183220/444
9	TO	70/363	333330/46881	-	TO	TO
11	-	230/1010	TO	-	-	-
13	-	700/2594	-	-	-	-
Ring 2	20/15	20/7	30/15	80/8	80/7	80/8
3	20/59	20/11	70/59	1500/41	500/8	920/19
4	30/232	20/15	350/232	51310/77	6370/11	19300/40
5	160/973	20/19	2270/973	TO	283780/16	TO
6	880/4057	30/23	14250/4057	-	TO	-
7	5940/16299	40/27	83360/16299	-	-	-
8	39120/64657	40/31	510850/64657	-	-	-
9	251370/261394	50/35	TO	-	-	-
10	TO	70/39	-	-	-	-

Table 5. Results for $POND$ for conformant Rovers, Logistics, BT, BTC, Cube Center, and Ring. The data is Total Time (ms)/# Expanded Nodes, “TO” indicates a time out and “-” indicates no attempt.

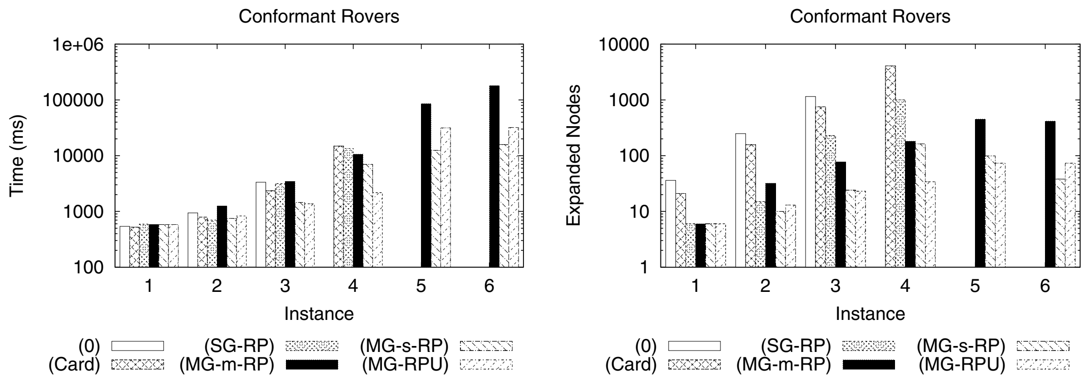


Figure 15. Total Time (ms) on left and Expanded Nodes on right in Conformant Rovers domain.

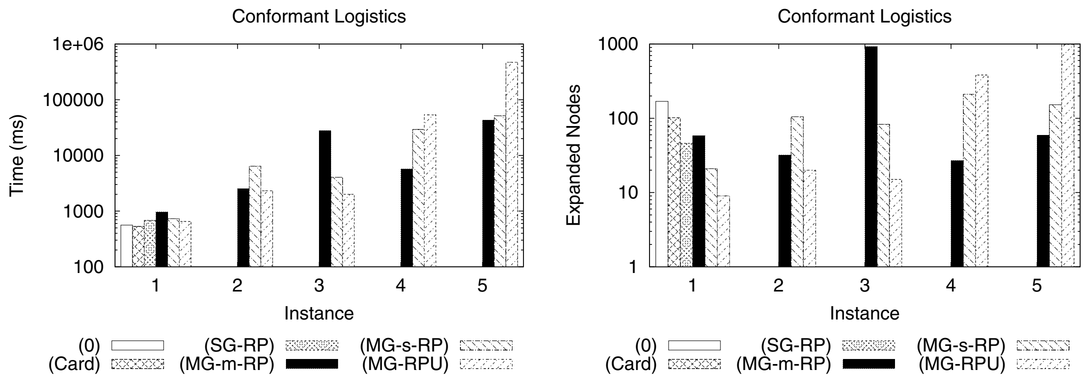


Figure 16. Total Time (ms) on left and Expanded Nodes on right in Conformant Logistics domain.

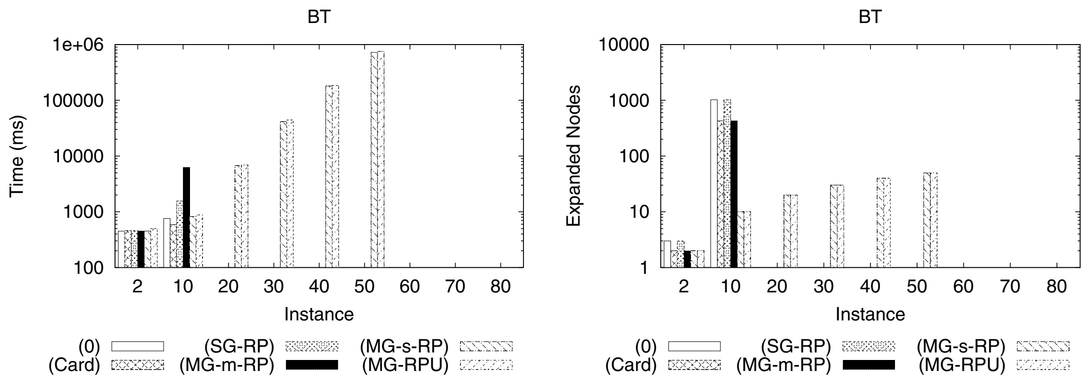


Figure 17. Total Time (ms) on left and Expanded Nodes on right in BT domain.

in the Rovers, Cube Center, and Ring domains, but fails to scale on the BT, BTC, and Logistics domains. Approximating a fully observable plan with the single graph relaxed plan is reasonable when plans for achieving the goal from each world have high positive interaction, as in Rovers,

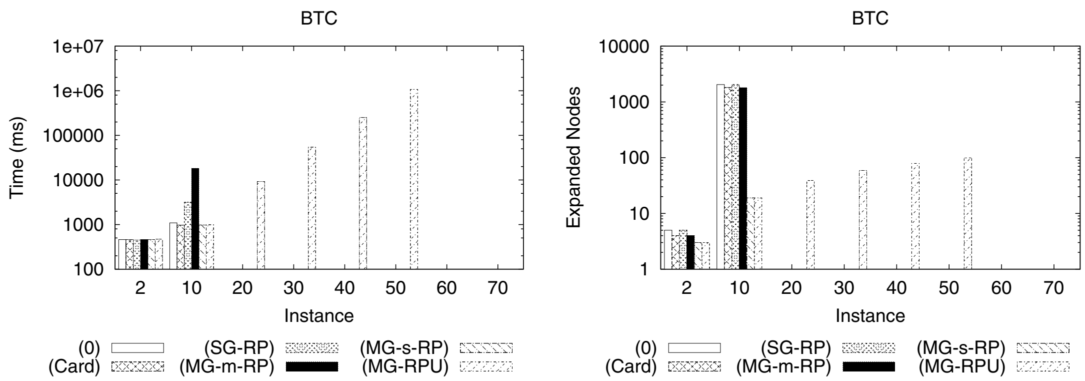


Figure 18. Total Time (ms) on left and Expanded Nodes on right in BTC domain.

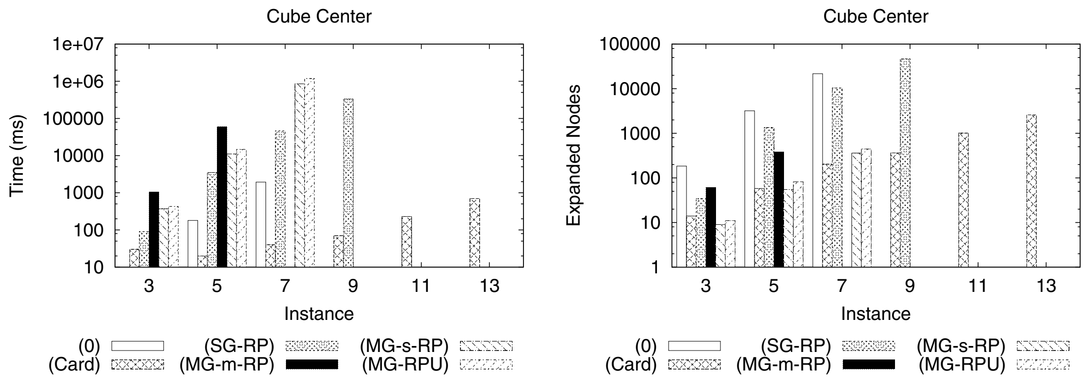


Figure 19. Total Time (ms) on left and Expanded Nodes on right in Cube Center domain.

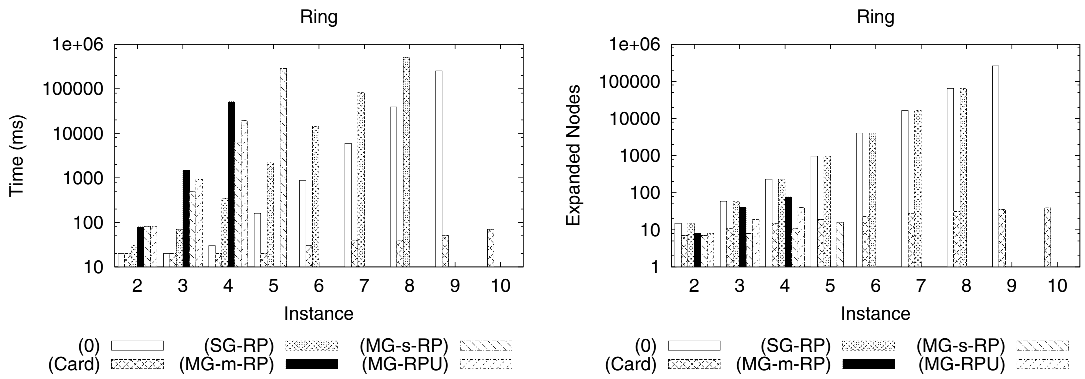


Figure 20. Total Time (ms) on left and Expanded Nodes on right in Ring domain.

Cube Center, and Ring. However, without high positive interaction, as in BT, BTC, and Logistics, the heuristic degrades quickly when the number of initial worlds increases.

With multiple planning graphs, $POND$ is able to perform better in the BT, BTC, Rovers, and Logistics domains, but takes quite a bit of time in the Cube Center and Ring domains. In Rovers, capturing distance estimates for individual worlds and aggregating them by some means tends to be better than aggregating worlds and computing a single distance estimate (as in SG^U). In BT and BTC, the total time increases quickly because the number of planning graphs and the number of relaxed plans for every search node increase greatly as problems become larger.

It is possible to compare the choices for aggregating the distance measures with the MG relaxed plans. Taking the maximum cost of the relaxed plans, h^{MG}_{m-RP}, in assuming positive interaction among worlds is useful in Logistics and Rovers, but loses the independence of worlds in the Ring, Cube Center, BT, and BTC domains. However, taking the summation of the relaxed plan values for different worlds, h^{MG}_{s-RP} is able to capture the independence in the BT domain. The summation does not help $POND$ in the BTC domain (Figure 18) because it overestimates the heuristic value for some nodes by counting the Flush action once for each world when it in fact only needs to be done once (i.e., it misses positive interaction). Finally, using the h^{MG}_{RPU} heuristic works well in every domain, aside from the cost of computing multiple graph heuristics (which is high in Ring and Cube Center), because it accounts for both positive interaction and independence by taking the overlap of relaxed plans. However, there is still a need to capture more possible worlds in a cheaper fashion.

The performance of $POND$ in the conditional domains exhibits similar trends to the conformant domains, with a few exceptions. Like the conformant domains, the MG relaxed plans tend to outperform the SG relaxed plan. Unlike the conformant domains, The h^{MG}_{m-RP} performs much better in BTS and BTCS over BT and BTC partly because the conditional plans have a lower average cost. The h_{card} heuristic does better in BTS and BTCS over BT and BTC because the belief states actually decrease in size when they are partitioned by sensory actions.

Problem	h_0	h_{card}	h^{SG}_{RP}	h^{MG}_{m-RP}	h^{MG}_{s-RP}	h^{MG}_{RPU}
Rovers 1	550/36	480/21	580/6	570/6	570/6	580/6
2	1030/262	550/36	780/15	760/14	710/12	730/12
3	1700/467	590/48	3930/248	830/15	830/15	910/17
4	5230/1321	620/58	6760/387	1020/20	1040/21	1070/21
5	TO	TO	TO	16360/175	11100/232	12810/209
6	-	-	-	31870/173	24840/159	30250/198
Logistics 1	530/118	TO	740/46	580/10	570/10	600/10
2	TO	-	TO	1630/30	1300/36	1360/36
3	-	-	-	1360/20	1250/19	1290/19
4	-	-	-	4230/59	3820/57	3940/57
5	-	-	-	27370/183	19620/178	20040/178
BT 2	460/5	460/3	450/3	460/3	450/3	470/3
10	TO	470/19	111260/7197	970/19	970/19	1020/19
20	-	510/39	TO	9070/39	9060/39	9380/39
30	-	620/59	-	52410/59	52210/59	55750/59
40	-	850/79	-	207890/79	206830/79	233720/79
50	-	1310/99	-	726490/99	719000/99	TO
60	-	2240/119	-	TO	TO	-
70	-	24230/139	-	-	-	-
80	-	45270/159	-	-	-	-
BTC 2	450/6	460/3	470/5	470/3	460/3	470/3
10	TO	480/19	271410/10842	1150/19	1140/19	1200/19
20	-	510/39	TO	11520/39	TO	11610/39
30	-	660/59	-	62060/59	-	64290/59
40	-	970/79	-	251850/79	-	274610/79
50	-	1860/99	-	941220/99	-	TO
60	-	4010/119	-	TO	-	-
70	-	7580/139	-	-	-	-

Table 6. Results for $POND$ for conditional Rovers, Logistics, BTS, BTCS. The data is Total Time (ms)/# Expanded Nodes, “TO” indicates a time out (20 minutes) and “-” indicates no attempt.

5.2. Sampling Worlds. The evaluations to this point have considered the effectiveness of different heuristics, each computed with respect to all possible worlds of a belief state. While it is more informed to use as many of the possible worlds as possible, one can reduce computation cost and hope to still get reasonable heuristics by considering a subset of the worlds. The scheme for considering subsets of worlds in the heuristics is to sample a single world (SG^1), or sample a given percentage of the worlds and build multiple graphs, or the LUG.

With these sampling approaches, I use the h^{SG}_{RP}, h^{MG}_{RPU}, and h^{LUG}_{RP} relaxed plans. I build the MG and LUG for 10%, 30%, 50%, 70%, and 90% of the worlds in each belief state, sampled

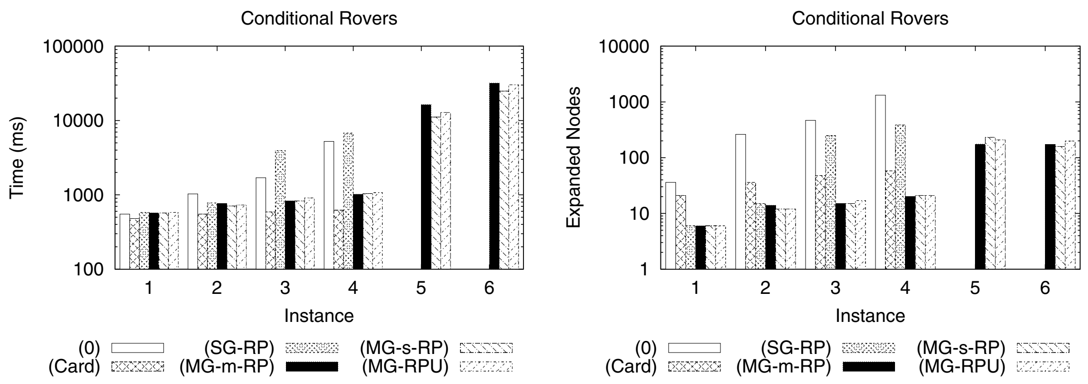


Figure 21. Total Time (ms) on left and Expanded Nodes on right in Conditional Rovers domain.

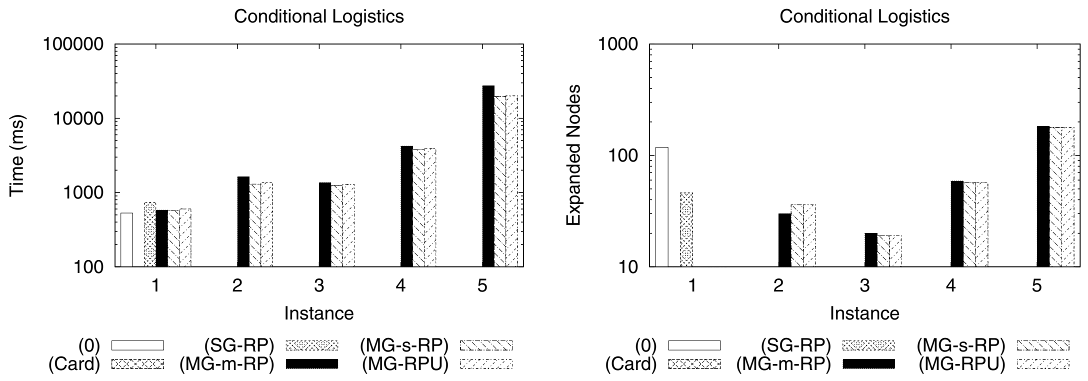


Figure 22. Total Time (ms) on left and Expanded Nodes on right in Conditional Logistics domain.

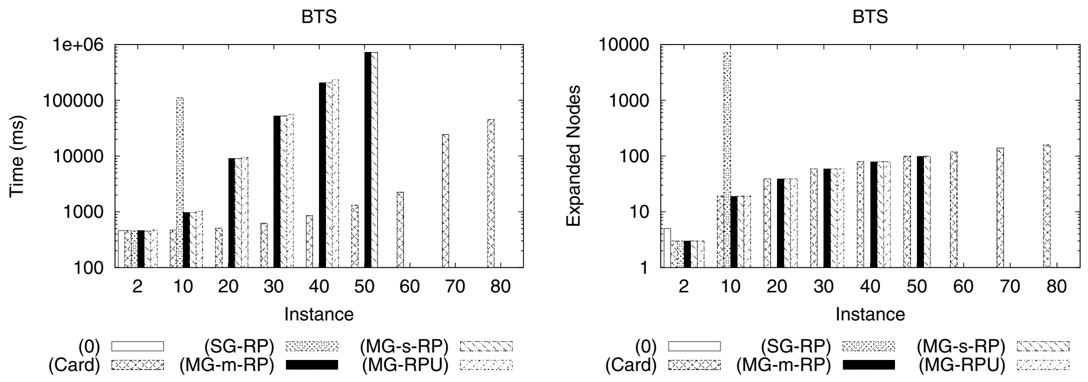


Figure 23. Total Time (ms) on left and Expanded Nodes on right in BTS domain.

uniformly and randomly. In Figure 25, I show the total time taken (hours) to solve every problem in the test set (79 problems over 10 domains). Each unsolved problem contributed 20 minutes to the total time. For comparison I show the previously mentioned heuristics: h^{SG}_{RP} computed on a unioned

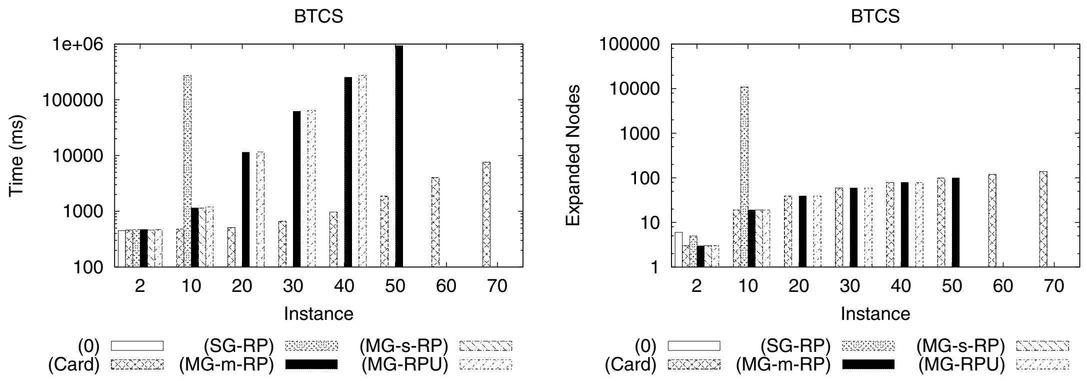


Figure 24. Total Time (ms) on left and Expanded Nodes on right in BTCS domain.

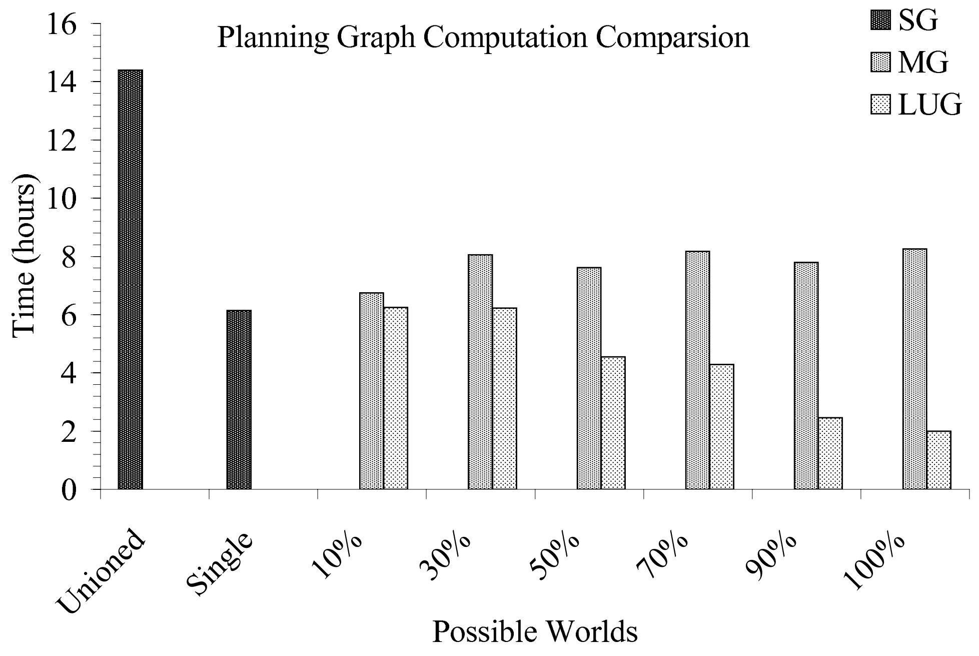


Figure 25. Total Time (hours) for $POND$ to solve all conformant and conditional problems when sampling worlds to use in heuristic computation.

single graph SG^U, denoted as “Unioned” compared to the sampled single graph SG^1 denoted as “Single”, and h^{MG}_{RPU} and h^{LUG}_{RP} computed for all worlds denoted as “100%”. The total time for any heuristic that samples worlds is averaged over ten runs.

There are two major points to see in Figure 25. First, the h^{SG}_{RP} heuristic is much more effective when computed on SG^1 versus SG^U. This is because the SG^1 is less optimistic. It builds a planning graph for a real world state, as opposed to the union of propositions in all possible world states, as in SG^U. Respecting state boundaries and considering only a single state is better than ignoring state boundaries to naively consider all possible states. However, as seen with the MG and LUG heuristics, respecting state boundaries and considering several states can be much better, bringing the second point.

There is very different performance when using more possible worlds to build multiple graphs compared to the LUG. It is better using fewer worlds when building multiple graphs because they can become very costly as the number of worlds increases. Multiple graph performance does not degrade considerably as the number of worlds increases. The number of problems that it can solve remains more or less constant. Otherwise the number of unsolved problems would increase and dominate the total time. In contrast, performance improves with more possible worlds when using the LUG. Using more possible worlds to compute heuristics is a good idea, but it takes a more efficient substrate to exploit them.

5.3. Accuracy. The heuristics that account for overlap in the possible worlds should be more accurate than the heuristics that make an assumption of full positive interaction or full independence. To check this intuition, I compare the heuristic estimates for the distance between the initial belief state and the goal for all the heuristics used in conformant problems. Figure 26 shows the ratio of the heuristic estimate for $h(b_I, G)$ to the optimal serial plan length $h^*(b_I, G)$ in several problems. The points below the line (where the ratio is one) are under-estimates, and those above are over-estimates. Some of the problem instances are not shown because no optimal plan length is known.

In all domains, the h_{RP}^{LUG} and h_{RPU}^{MG} heuristics are very close to h^*, confirming the intuitions.

Interestingly, h^{MG}_{s-RP} and h^{MG}_{m-RP} are both close to h^* in Rovers and Logistics; whereas the former is close in the BT and BTC problems, and the latter is close in CubeCenter and Ring. As expected, assuming independence (using summation) tends to over-estimate, and assuming positive interaction (using maximization) tends to under-estimate. The h^{SG}_{RP} heuristic tends to under-estimate, and in some cases (CubeCenter and Ring) gives a value of zero (because there is an initial state that satisfies the goal). The h_{card} heuristic is accurate in BT and BTC, under-estimates in Rovers and Logistics, and over-estimates in Cube Center and Ring.

The accuracy of heuristics is in some cases disconnected from their run time performance. For instance h_{card} highly overestimates in Ring and Cube Center, but does well because the domains exhibit special structure and the heuristic is fast to compute. On the other hand, h^{LUG}_{RP} and h^{MG}_{RPU} are very accurate in many domains, but suffer in Ring and Cube Center because they can be costly to compute. We will see in chapter 7 how it is possible to further improve performance on these domains by reusing the *LUG* between search nodes that have many states in common.

5.4. Empirical Evaluation Conclusions. In comparing ways to aggregate state distance measures to compute belief state distances, I found that measuring no interaction as in single graph heuristics tends to poorly guide planners, measuring independence and positive interaction of worlds works well in specific domains, and measuring overlap (i.e., a combination of positive interaction and independence) tends to work well in a large variety of instances. By studying the accuracy of our heuristics I found that in some cases the most accurate were not the most effective. I did however find that the most accurate did best over the most cases. The following list summarizes the findings:

- Of the planning graph heuristics presented, relaxed plans that take into account the overlap of individual plans between states of the source and destination belief states are the most

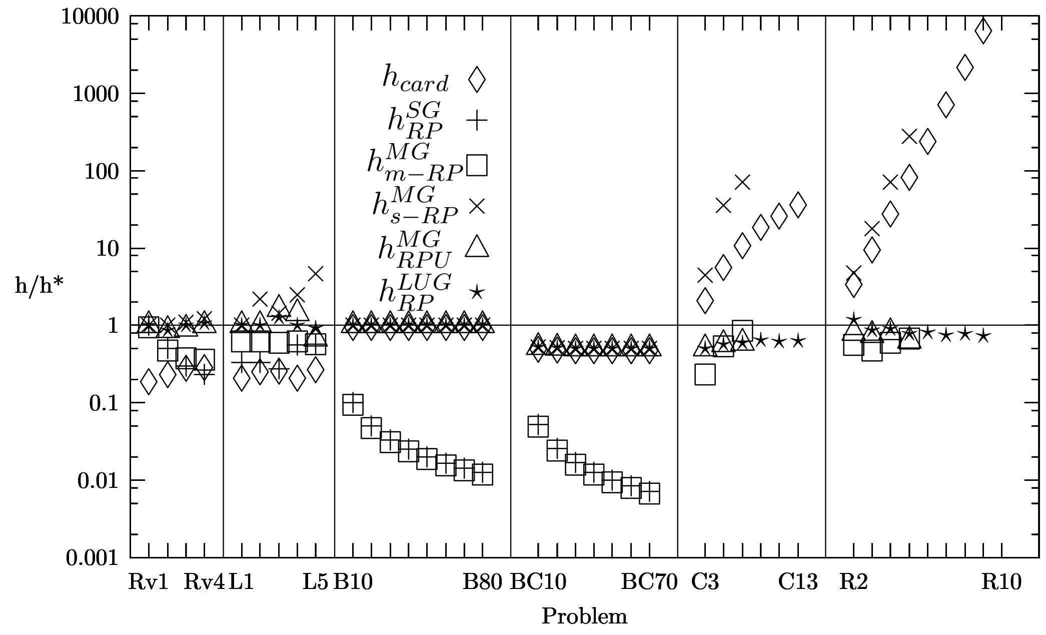


Figure 26. Ratio of heuristic estimates for distance between b_I and G to optimal plan length. Rv = Rovers, L = Logistics, B = BT, BC = BTC, C = Cube Center, R = Ring.

accurate and tend to perform well across many domains.

- Sampling possible worlds to construct planning graphs does reduce computational cost, but considering more worlds by exploiting planning graph structure common to possible worlds (as in the LUG), can be more efficient and informed.

- The heuristics help our conditional planner, $POND$, to scale up in conditional domains, despite the fact that the heuristic computation does not model action observations.

6. Related Work

Many works have described approaches for formulating conformant and conditional non-deterministic planning. These include extensions of GraphPlan (Blum and Furst, 1995) to conformant planning (Smith and Weld, 1998; Kurien *et al.*, 2002) and conditional planning (Weld *et* *al.*, 1998). Others have used variants of satisfiability (Castellini *et al.*, 2001; Palacios and Geffner, 2005), or logic programming (Morales *et al.*, 2007).

Perhaps the most popular set of approaches use search in the space of belief states (Genesereth and Nourbakhsh, 1993; Bonet and Geffner, 2000; Bertoli *et al.*, 2001; Bertoli and Cimatti, 2002; Rintanen, 2002; Tuan *et al.*, 2004; Brafman and Hoffmann, 2004), which this work also employs. Is is also possible to search in the space of states by transforming problems to classical planning (Palacios and Geffner, 2006).

7. Conclusion

With the intent of establishing a basis for belief state distance estimates, I have:

- Discussed how heuristic measures can aggregate state distance measures to capture positive interaction, negative interaction, independence, and overlap.
- Found that distance based heuristics for belief space search help control conformant and conditional plan length because, as opposed to cardinality, the heuristics model desirable plan quality metrics.
- Shown how to compute such heuristic measures on planning graphs and provided empirical comparisons of these measures.

The intent in this chapter is to provide a formal basis for measuring the distance between belief states in terms of underlying state distances. I investigated several ways to aggregate the state distances to reflect various assumptions about the interaction of state to state trajectories. The best of these measures turned out to measure both positive interaction and independence in what I call overlap. I saw that planners using this notion of overlap tend to do well across a large variety of domains and tend to have more accurate heuristics.

CHAPTER 4

LABELED PLANNING GRAPHS

This chapter describes labeled planning graphs and how they can be used to compute relaxed plans to estimate belief state distance measures. It includes a description of labeled planning graphs (with a worked example), an empirical evaluation, and a discussion of related work. This chapter focusses on non-deterministic conformant and conditional planning with deterministic actions.

1. Introduction

In the previous chapter, I identified how the multiple graph technique for computing belief state distance measures can compute heuristics that aggregate the costs of multiple worlds. However, this came with the disadvantages of computing some redundant information in different graphs (c.f. Figure 14) and using every graph to compute heuristics (c.f h^{MG}_{RPU}). The approach described in this chapter addresses these limitations by condensing the multiple planning graphs to a single planning graph, called a labeled uncertainty graph (LUG). The idea is to implicitly represent multiple planning graphs by collapsing the graph connectivity into one planning graph, but use annotations, called labels (ℓ), to retain information about multiple worlds. While it is possible to construct the LUG by generating each of the multiple graphs and taking their union, instead I define a direct construction procedure. The procedure starts similar to the unioned single planning graph (SG^U) by constructing an initial layer of all propositions in each state of a belief state. The difference with the LUG is that it can prevent loss of information about multiple worlds by recording in a label for each proposition which of the possible worlds is relevant. I will discuss a few simple techniques used to propagate the labels through actions and effects to label subsequent proposition layers. Label propagation relies on expressing labels as propositional formulas and using standard propositional logic operations, but I will describe it using set logic. The end product is single planning graph with labels on all graph vertices; labels denote in which of the explicit multiple graphs (if they were built) a graph vertex is reached.

The LUG trades planning graph structure space for label storage space. Using BDDs (Bryant, 1986) to represent labels helps lower the storage requirements on labels. The worst-case complexity of the LUG is equivalent to the MG representation. The LUG's complexity savings is not realized when the projected possible worlds and the relevant actions for each are completely disjoint; however, this does not often appear in practice. The space savings comes in through two aspects: (1) redundant representation of actions and propositions is avoided, and (2) labels that facilitate non-redundant representation are stored as BDDs. A nice feature of BDDs in the package used (Somenzi, 1998) is that they efficiently represent many BDDs in a shared BDD that leverages common substructure. Hence, in practice the LUG contains the same information as MG with much lower construction and usage costs.

2. Labeled Uncertainty Graph Heuristics (LUG)

In this section I present how to construct the LUG and how to extract relaxed plans. I will use the same assumptions from the previous chapter, namely actions are deterministic, the model is non-deterministic, and observations are ignored in the heuristics for conditional planning problems.

2.1. Label Propagation. Like the single graph and multiple graphs, the LUG is based on the IPP (Koehler *et al.*, 1997) planning graph. I extend the single graph to capture multiple world causal support, as present in multiple graphs, by adding labels to the vertices of the action $\mathcal{A}$, effect $\mathcal{E}$, and proposition $\mathcal{P}$ layers, such that the **LUG** is a set of vertices and a label function: $LUG(b) = \langle(\mathcal{P}_0, \mathcal{A}_0, \mathcal{E}_0, ..., \mathcal{A}_{k-1}, \mathcal{E}_{k-1}, \mathcal{P}_k), \ell\rangle$. I denote the label of a proposition p in level k as $\ell(k, p)$, which will be described with the short-hand $\ell_k(p)$. A label is a set of states (also in belief state b) from which a graph vertex is (optimistically) *reachable*. A proposition p is reachable from a set of states, described by b, after k levels, if $\ell_k(p) = b$. For instance, I can say that `comm(soil)` is reachable after three levels from b_I if the LUG has a proposition layer $\mathcal{P}_2$ containing `comm(soil)` and ℓ_2(`comm(soil)`) = b_I, meaning that the states where `comm(soil)` holds after two levels are exactly the set of states in b_I.

The intuitive definition of the LUG is a planning graph skeleton, that represents causal relations, over which propagated labels indicate specific possible world support. I show the LUG for the conformant rovers problem in Figure 27. There is a set of shaded circles above each proposition and action to represent labels (please ignore the variations in circle size for now). The black circles represent the planning graph for the first state s_α = {`at(alpha), avail(soil, alpha)`}, the grey circles are for the state s_β = {`at(alpha), avail(soil, beta)`}, and the white circles are for the state s_γ = {`at(alpha), avail(soil, gamma)`}. The initial proposition layer has `at(alpha)` labeled with every color because it holds in each of the states, and the other propositions are labeled to indicate their respective states.

Constructing the graph skeleton largely follows traditional planning graph semantics, and label propagation relies on a few simple rules. Each initial layer proposition is labeled, to indicate the states of b in which it holds. An action is labeled to indicate all states that co-achieve its execution preconditions, which is the intersection of the labels of its execution preconditions. An effect is labeled to indicate all worlds that co-achieve its antecedent propositions and its action's execution preconditions, which is the intersection of the labels of its antecedent propositions and the label of its associated action. Finally, propositions are labeled to indicate all worlds where they are given as an effect, which is the union over all labels of effects in the previous level that affect the proposition. I now describe label propagation in more detail and work through the conformant rover example.

Initial Proposition Layer: The $LUG(b)$ has an initial layer consisting of every proposition with a non empty label. In the initial layer the label $\ell_0(p)$ of each proposition p is identical to

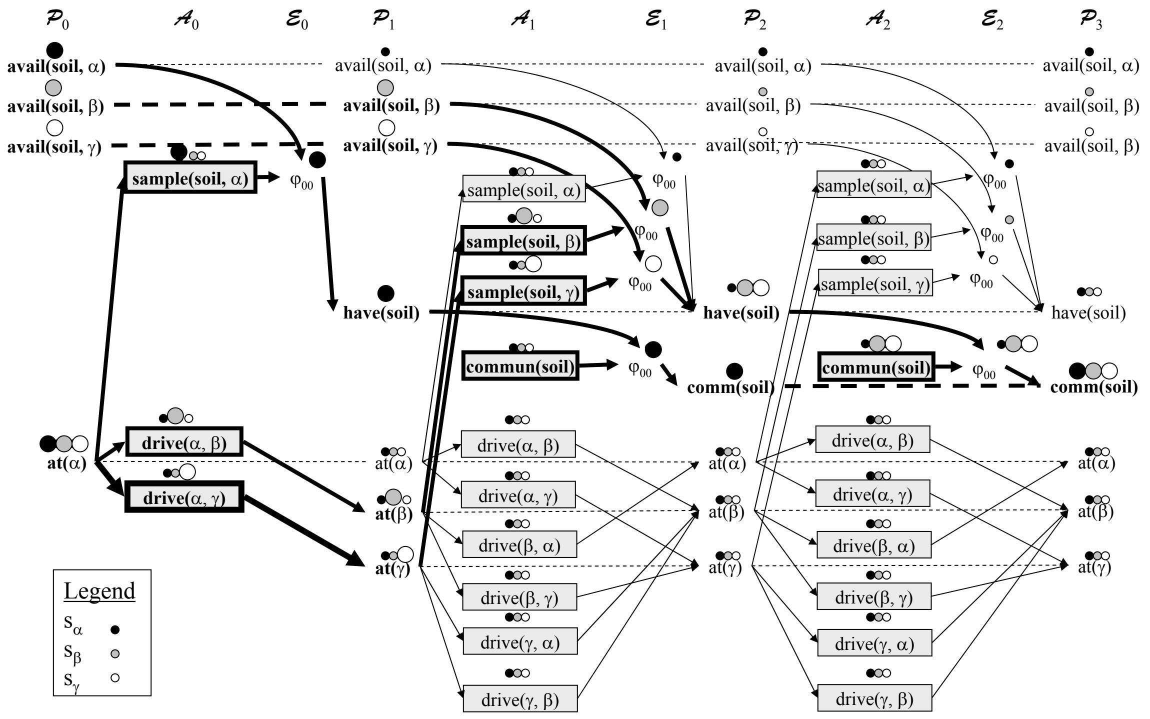


Figure 27. The $LUG(b_I)$ of the conformant rover problem. The relaxed plan for h^{LUG}_{RP} is shown in bold.

$\ell_o(p) = \bigcup_{p \in s, s \in b} s$

representing the states of b in which p holds.

The labels for the initial layer propositions are propagated through actions and effects to label the next proposition layer, as I will describe shortly. Propagation continues until either no label of any proposition changes between layers, a condition referred to as level off, or each goal proposition has its label contain all states in b (meaning the goal is reachable).

The $LUG(b_I)$ of the conformant rovers problem, shown in Figure 27, has the initial proposition layer:

$$\mathcal{P}_0 = \{\texttt{avail(soil, alpha), avail(soil, beta),}$$
$$\texttt{avail(soil, gamma), at(alpha)}\}$$

$$\begin{aligned}
\ell_0(\texttt{avail(soil, alpha)}) &= \{s_\alpha\}\\
\ell_0(\texttt{avail(soil, beta)}) &= \{s_\beta\}\\
\ell_0(\texttt{avail(soil, gamma)}) &= \{s_\gamma\}\\
\ell_0(\texttt{at(alpha)}) &= \{s_\alpha, s_\beta, s_\gamma\}
\end{aligned}$$

Notice that each `avail` proposition has a label indicating the respective initial state in which it holds, and `at(alpha)` is labeled by every state in b_I.

Action Layer: Once the previous proposition layer $\mathcal{P}_k$ is computed, it is possible to find the action layer $\mathcal{A}_k$. $\mathcal{A}_k$ contains causative actions from the action set A, plus proposition persistence. An action is included in $\mathcal{A}_k$ if its label is non empty. The label of an action at level k, is equivalent to the intersection of the labels of its execution precondition proposition labels:

$$\ell_k(a) = \bigcap_{p \in \rho_e(a)} \ell_k(p)$$

The zeroth action layer for conformant rovers problem, is:

$$\begin{aligned}
\mathcal{A}_0 = \; &\{\texttt{sample(soil, alpha), drive(alpha, beta),}\\
&\texttt{drive(alpha, gamma),avail(soil, alpha)}_p\texttt{,}\\
&\texttt{avail(soil, beta)}_p\texttt{, avail(soil, gamma)}_p\texttt{,}\\
&\texttt{at(alpha)}_p\}
\end{aligned}$$

$$\begin{aligned}
\ell_0(\texttt{sample(soil, alpha)}) = \ell_0(\texttt{drive(alpha, beta)}) &=\\
\ell_0(\texttt{drive(alpha, gamma)}) = \ell_0(\texttt{at(alpha)}_p) &= \{s_\alpha, s_\beta, s_\gamma\}\\
\ell_0(\texttt{avail(soil, alpha)}_p) &= \{s_\alpha\}\\
\ell_0(\texttt{avail(soil, beta)}_p) &= \{s_\beta\}\\
\ell_0(\texttt{avail(soil, gamma)}_p) &= \{s_\gamma\}
\end{aligned}$$

Each proposition persistence has a label identical to the label of the corresponding proposition from the previous proposition layer. The `sample` and both `drive` actions have labels identical to b_I because their enabling precondition `at(alpha)` has the same label. While the actions are labeled with every state in b_I, it does not meant their effects will have the same label.

Effect Layer: The effect layer $\mathcal{E}_k$ depends both on the proposition layer $\mathcal{P}_k$ and action layer $\mathcal{A}_k$. $\mathcal{E}_k$ contains an effect $\varphi_{ij}(a)$ if the effect has a non empty label. Because both the action and the effect must be applicable in the same world, the label of the effect at level k is the intersection of the label of the associated action with the intersection of the labels of the antecedent propositions:

$$\ell_k(\varphi_{ij}(a)) = \ell_k(a) \cap \bigcap_{p \in \rho_{ij}(a)} \ell_k(p)$$

The zeroth effect layer for the example, is:

$$\begin{aligned}\mathcal{E}_0 = \ & \{\varphi_{00}(\texttt{sample(soil, alpha)}), \varphi_{00}(\texttt{drive(alpha, beta)}),\\ & \varphi_{00}(\texttt{drive(alpha, gamma)}), \varphi_{00}(\texttt{avail(soil, alpha)}_p),\\ & \varphi_{00}(\texttt{avail(soil, beta)}_p), \varphi_{00}(\texttt{avail(soil, gamma)}_p),\\ & \varphi_{00}(\texttt{at(alpha)}_p)\}\end{aligned}$$

$$\begin{aligned}&\ell_0(\varphi_{00}(\texttt{drive(alpha, beta)})) = \ell_0(\varphi_{00}(\texttt{drive(alpha, gamma)})) = \\ &\ell_0(\varphi_{00}(\texttt{at(alpha)}_p)) &&= \{s_\alpha, s_\beta, s_\gamma\}\\ &\ell_0(\varphi_{00}(\texttt{sample(soil, alpha)})) = \ell_0(\varphi_{00}(\texttt{avail(soil, alpha)}_p)) &&= \{s_\alpha\}\\ &\ell_0(\varphi_{00}(\texttt{avail(soil, beta)}_p)) &&= \{s_\beta\},\\ &\ell_0(\varphi_{00}(\texttt{avail(soil, gamma)}_p)) &&= \{s_\gamma\}\end{aligned}$$

Again, like the action layer, the effect of each proposition persistence has a label identical to the corresponding proposition in the previous proposition layer. The effect of each `drive` action has a label identical to the label of the associated action. The label of the effect of the `sample` action is different from the label of the action because the intersection of the antecedent proposition label and the action label is one state, $\{s_\alpha, s_\beta, s_\gamma\} \cap \{s_\alpha\} = \{s_\alpha\}$.

Proposition Layer: The proposition layer $\mathcal{P}_k$ depends on the previous effect layer $\mathcal{E}_{k-1}$, and contains only propositions with non empty labels. An effect $\varphi_{ij}(a) \in \mathcal{E}_{k-1}$ contributes to the label of a proposition p when the effect consequent contains the proposition p. The label of a proposition is the disjunction of the labels of each effect from the previous effect layer that gives the proposition:

$$\ell_k(p) = \bigcup_{\varphi_{ij}(a) \in \mathcal{E}_{k-1} : p \in \varepsilon_{ij}(a)} \ell_{k-1}(\varphi_{ij}(a))$$

The first proposition layer for the example is:

$$\mathcal{P}_1 = \mathcal{P}_0 \cup \{\texttt{at(beta), at(gamma), have(soil)}\}$$

$$\begin{array}{lcl}
\ell_1(\texttt{at(alpha)}) & = & \ell_0(\texttt{at(beta)}) & = & \\
\ell_1(\texttt{at(gamma)}) & & & = & \{s_\alpha, s_\beta, s_\gamma\} \\
\ell_1(\texttt{avail(soil, alpha)}) & = & \ell_0(\texttt{have(soil)}) & = & \{s_\alpha\} \\
\ell_1(\texttt{avail(soil, beta)}) & & & = & \{s_\beta\} \\
\ell_1(\texttt{avail(soil, gamma)}) & & & = & \{s_\gamma\}
\end{array}$$

This proposition layer is identical the initial proposition layer, except that `have(soil)` is now reachable from s_α because it is supported by the effect of the `sample(soil, alpha)` action.

Continuing to the level one action layer:

$$\begin{aligned}\mathcal{A}_1 = \mathcal{A}_0 \cup\ & \{\texttt{sample(soil, beta)},\ \texttt{sample(soil, gamma)},\\ & \texttt{commun(soil)},\ \texttt{drive(beta, alpha)},\\ & \texttt{drive(beta, gamma)},\ \texttt{drive(gamma, alpha)},\\ & \texttt{drive(gamma, beta)},\texttt{have(soil)}_p\}\end{aligned}$$

$$\begin{aligned}
\ell_1(\texttt{sample(soil, alpha)}) &= \ell_1(\texttt{drive(alpha, beta)}) &&= \\
\ell_1(\texttt{drive(alpha, gamma)}) &= \ell_1(\texttt{at(alpha)}_p) &&= \\
\ell_1(\texttt{sample(soil, beta)}) &= \ell_1(\texttt{sample(soil, gamma)}) &&= \\
\ell_1(\texttt{commun(soil)}) &= \ell_1(\texttt{drive(beta, alpha)}) &&= \\
\ell_1(\texttt{drive(beta, gamma)}) &= \ell_1(\texttt{drive(gamma, alpha)}) &&= \\
\ell_1(\texttt{drive(gamma, beta)}) &= \ell_1(\texttt{at(alpha)}_p) &&= \\
\ell_1(\texttt{at(beta)}_p) &= \ell_1(\texttt{at(gamma)}_p) &&= \{s_\alpha, s_\beta, s_\gamma\} \\
\ell_1(\texttt{avail(soil, alpha)}_p) &= \ell_1(\texttt{have(soil)}_p) &&= \{s_\alpha\} \\
\ell_1(\texttt{avail(soil, beta)}_p) & &&= \{s_\beta\} \\
\ell_1(\texttt{avail(soil, gamma)}_p) & &&= \{s_\gamma\}
\end{aligned}$$

This action layer has new actions: the remaining `sample` actions, the `commun(soil)` action, and several `drive` actions. Each of the new actions is applicable in all possible worlds.

The level one effect layer is:

$$\begin{aligned}\mathcal{E}_1 = \mathcal{E}_0 \cup\ & \{\varphi_{00}(\texttt{sample(soil, beta)}), \varphi_{00}(\texttt{sample(soil, gamma)}),\\ & \varphi_{00}(\texttt{drive(beta, alpha)}), \varphi_{00}(\texttt{drive(beta, gamma)}),\\ & \varphi_{00}(\texttt{drive(gamma, alpha)}), \varphi_{00}(\texttt{drive(gamma, beta)}),\\ & \varphi_{00}(\texttt{have(soil)}_p), \varphi_{00}(\texttt{at(beta)}_p),\\ & \varphi_{00}(\texttt{at(gamma)}_p), \varphi_{00}(\texttt{commun(soil)})\}\end{aligned}$$

$$\begin{aligned}
\ell_1(\varphi_{00}(\texttt{drive(alpha, beta)})) &= \ell_1(\varphi_{00}(\texttt{drive(alpha, gamma)})) =\\
\ell_1(\varphi_{00}(\texttt{drive(beta, alpha)})) &= \ell_1(\varphi_{00}(\texttt{drive(beta, gamma)})) =\\
\ell_1(\varphi_{00}(\texttt{drive(gamma, alpha)})) &= \ell_1(\varphi_{00}(\texttt{drive(gamma, beta)})) =\\
\ell_1(\varphi_{00}(\texttt{at(beta)}_p)) &= \ell_1(\varphi_{00}(\texttt{at(gamma)}_p)) =\\
\ell_1(\varphi_{00}(\texttt{at(alpha)}_p)) &= \{s_\alpha, s_\beta, s_\gamma\}\\
\ell_1(\varphi_{00}(\texttt{sample(soil, alpha)})) &= \ell_1(\varphi_{00}(\texttt{commun(soil)})) =\\
\ell_1(\varphi_{00}(\texttt{have(soil)}_p)) &= \ell_1(\varphi_{00}(\texttt{avail(soil, alpha)}_p)) =\\
\ell_1(\varphi_{00}(\texttt{commun(soil)})) &= \{s_\alpha\}\\
\ell_1(\varphi_{00}(\texttt{sample(soil, beta)})) &= \ell_1(\varphi_{00}(\texttt{avail(soil, beta)}_p)) = \{s_\beta\}\\
\ell_1(\varphi_{00}(\texttt{sample(soil, gamma)})) &= \ell_1(\varphi_{00}(\texttt{avail(soil, gamma)}_p)) = \{s_\gamma\}
\end{aligned}$$

The conditional effects of the two new `sample` actions have labels to indicate the states that can reach them. The effect of the `commun(soil)` is also now reachable from state s_α. There are also several new effects for additional `drive` actions.

The level two proposition layer is as follows:

$$\mathcal{P}_2 = \mathcal{P}_1 \cup \{\texttt{comm(soil)}\}$$

$$\begin{aligned}
\ell_2(\texttt{at(alpha)}) &= \ell_2(\texttt{at(beta)}) =\\
\ell_2(\texttt{at(gamma)}) &= \ell_2(\texttt{have(soil)}) = \{s_\alpha, s_\beta, s_\gamma\}\\
\ell_2(\texttt{avail(soil, alpha)}) &= \ell_2(\texttt{comm(soil)}) = \{s_\alpha\}\\
\ell_2(\texttt{avail(soil, beta)}) &= \{s_\beta\}\\
\ell_2(\texttt{avail(soil, gamma)}) &= \{s_\gamma\}
\end{aligned}$$

The `have(soil)` proposition is now reachable by every state because it is supported by the effect of the three `sample` actions, each supporting from a different state. The goal proposition `commun(soil)` is reachable from one state. All other propositions are reachable from the same states as the previous proposition layer.

Continuing to the level two action layer:

$$\mathcal{A}_2 = \mathcal{A}_1 \cup \{\texttt{comm(soil)}_p\}$$

$$\begin{array}{lclcl}
\ell_2(\texttt{sample(soil, alpha)}) & = & \ell_2(\texttt{drive(alpha, beta)}) & = & \\
\ell_2(\texttt{drive(alpha, gamma)}) & = & \ell_2(\texttt{at(alpha)}_p) & = & \\
\ell_2(\texttt{sample(soil, beta)}) & = & \ell_2(\texttt{sample(soil, gamma)}) & = & \\
\ell_2(\texttt{commun(soil)}) & = & \ell_2(\texttt{drive(beta, alpha)}) & = & \\
\ell_2(\texttt{drive(beta, gamma)}) & = & \ell_2(\texttt{drive(gamma, alpha)}) & = & \\
\ell_2(\texttt{drive(gamma, beta)}) & = & \ell_2(\texttt{at(alpha)}_p) & = & \\
\ell_2(\texttt{at(beta)}_p) & = & \ell_2(\texttt{at(gamma)}_p) & = & \{s_\alpha, s_\beta, s_\gamma\} \\
\ell_2(\texttt{avail(soil, alpha)}_p) & = & \ell_2(\texttt{have(soil)}_p) & = & \{s_\alpha\} \\
\ell_2(\texttt{avail(soil, beta)}_p) & & & = & \{s_\beta\} \\
\ell_2(\texttt{avail(soil, gamma)}_p) & & & = & \{s_\gamma\} \\
\ell_2(\texttt{comm(soil)}_p) & & & = & \{s_\alpha\}
\end{array}$$

This action layer is exactly the same as the previous action layer, but includes a new persistence for `comm(soil)`.

The level two effect layer is:

$$\mathcal{E}_2 = \mathcal{E}_1$$

$$
\begin{aligned}
\ell_2(\varphi_{00}(\texttt{drive(alpha, beta)})) &= \ell_2(\varphi_{00}(\texttt{drive(alpha, gamma)})) = \\
\ell_2(\varphi_{00}(\texttt{drive(beta, alpha)})) &= \ell_2(\varphi_{00}(\texttt{drive(beta, gamma)})) = \\
\ell_2(\varphi_{00}(\texttt{drive(gamma, alpha)})) &= \ell_2(\varphi_{00}(\texttt{drive(gamma, beta)})) = \\
\ell_2(\varphi_{00}(\texttt{at(beta)}_p)) &= \ell_2(\varphi_{00}(\texttt{at(gamma)}_p)) = \\
\ell_2(\varphi_{00}(\texttt{at(alpha)}_p)) &= \ell_2(\varphi_{00}(\texttt{commun(soil)})) = \\
\ell_2(\varphi_{00}(\texttt{have(soil)}_p)) &= \{s_\alpha, s_\beta, s_\gamma\} \\
\ell_2(\varphi_{00}(\texttt{sample(soil, alpha)})) &= \ell_2(\varphi_{00}(\texttt{avail(soil, alpha)}_p)) = \{s_\alpha\} \\
\ell_2(\varphi_{00}(\texttt{sample(soil, beta)})) &= \ell_2(\varphi_{00}(\texttt{avail(soil, beta)}_p)) = \{s_\beta\} \\
\ell_2(\varphi_{00}(\texttt{sample(soil, gamma)})) &= \ell_2(\varphi_{00}(\texttt{avail(soil, gamma)}_p)) = \{s_\gamma\}
\end{aligned}
$$

The conditional effect of `commun(soil)` is now reachable from all possible worlds, meaning the next proposition layer will have the goal reachable from all possible worlds.

The level three proposition layer is as follows:

$$\mathcal{P}_3 = \mathcal{P}_2$$

$$
\begin{aligned}
\ell_0(\texttt{at(alpha)}) &= \ell_0(\texttt{at(beta)}) = \\
\ell_0(\texttt{at(gamma)}) &= \ell_0(\texttt{have(soil)}) = \\
\ell_0(\texttt{comm(soil)}) &= \{s_\alpha, s_\beta, s_\gamma\} \\
\ell_0(\texttt{avail(soil, alpha)}) &= \{s_\alpha\} \\
\ell_0(\texttt{avail(soil, beta)}) &= \{s_\beta\} \\
\ell_0(\texttt{avail(soil, gamma)}) &= \{s_\gamma\}
\end{aligned}
$$

The `comm(soil)` proposition is now reachable by every state because it is supported by the effect of the `commun(soil)` action, which is labeled with each state. All other propositions are reachable from the same states as the previous proposition layer. Construction can stop here because the goal is reachable from each state in b_I. However, level off occurs at the next level because there is no change in the labels of the propositions.

2.2. LUG **Heuristics.** The heuristics computed on the LUG can capture measures similar to the MG heuristics, but there exists a new opportunity to make use of labels to improve heuristic computation efficiency. A single planning graph is sufficient if there is no state aggregation being measured, so I do not mention such measures for the LUG. I proceed with aggregation techniques.

Positive Interaction Aggregation: Unlike MG heuristics, I do not compute positive interaction based relaxed plans on the LUG. The MG approach to measure positive interaction across plans for each state in a belief state is to compute multiple relaxed plans and take their maximum value. To get the same measure on the LUG one would still need to extract multiple relaxed plans, the situation avoided by using the LUG. While the graph construction overhead may be lowered by using the LUG, the heuristic computation could take too long. Hence, I do not compute relaxed plans on the LUG to measure positive interaction alone, but I do compute relaxed plans that measure overlap (which measures positive interaction).

Independence Aggregation: Like positive interaction aggregation, one needs a relaxed plan for every state in the projected belief state to find the summation of the costs. Hence, I do not compute relaxed plans that assume independence.

State Overlap Aggregation: A relaxed plan extracted from the LUG to get the h_{RP}^{LUG} heuristic resembles the unioned relaxed plan in the h_{RPU}^{MG} heuristic. Recall that the h_{RPU}^{MG} heuristic extracts a relaxed plan from each of the multiple planning graphs (one for each possible world) and unions the set of actions chosen at each level in each of the relaxed plans. The LUG relaxed plan heuristic is similar in that it counts actions that have positive interaction in multiple worlds only once and accounts for independent actions that are used in subsets of the possible worlds. The advantage of

h^{LUG}_{RP} is that it is possible to find these actions with a single pass on one planning graph.

The trade off is the cost of computing multiple relaxed plans and the cost of manipulating LUG labels to determine what lines of causal support are used in what worlds. In the relaxed plan the intent is to support the goal with every state in the projected belief state b, but in doing so it is necessary to track which states in b use which paths in the planning graph. A subgoal may have several different (and possibly overlapping) paths from the worlds in b.

A LUG relaxed plan is a subset of the planning graph layers and a label function defined as: $\langle(\mathcal{A}^{RP}_0, \mathcal{E}^{RP}_0, \mathcal{P}^{RP}_1, ..., \mathcal{A}^{RP}_{k-1}, \mathcal{E}^{RP}_{k-1}, \mathcal{P}^{RP}_k), \ell^{RP}\rangle$, where $\mathcal{A}^{RP}_r$ is a set of actions, $\mathcal{E}^{RP}_r$ is a set of effects, and $\mathcal{P}^{RP}_{r+1}$ is a set of propositions. The vertices of the layers are labeled to indicate the worlds of b where they are chosen for support. The relaxed plan is extracted from the level k where the goal propositions are first reachable by every state in b.

The labeled relaxed plan extraction is similar to the classical planning relaxed plan, but with one key difference. *A proposition may need support from more than one action because it needs support from several source states, each requiring a supporter.* Specifically, the relaxed plan tracks which of the source states require the inclusion of a particular action or proposition. Notice the circles in Figure 27 that are slightly larger than the others: these indicate the source states where the given action or proposition is needed. For instance, `at(gamma)` in $\mathcal{P}^{RP}_1$ is only used for the source state s_α. Supporting a proposition is a set cover problem where each action covers some set of states (indicated by its label). For example, `have(soil)` in $\mathcal{P}^{RP}_2$ needs support in the source states where it supports `commun(soil)`. There is no single action that supports it in all of these states. However, a subset of the supporting actions can support it by covering the required source states. For example, to cover `have(soil)` in $\mathcal{P}^{RP}_2$ with supporters, the relaxed plan includes the effect of `sample(soil, beta)` for support from state s_β, and the effect of `sample(soil, alpha)` for support from state s_γ. Using the notion of set cover, it easy to exploit positive interactions with a simple greedy heuristic (which can be used to bias action choice in line 7 of Figure 3): an action that supports in several of the explicit planning graphs is better than an action that supports in only one. I also bias relaxed plan extraction to use all persistence before using the number of worlds to bias effect selection.

For this example, the labeled relaxed plan gives a heuristic measure of 7, which is the same as the unioned relaxed plan extracted from multiple planning graphs. The relaxed plan, shown in bold in Figure 27, for b_I to reach G is listed as follows:

$$\begin{aligned}\mathcal{P}_0^{RP} = \ & \{\texttt{avail(soil, alpha)}, \texttt{avail(soil, beta)}, \\ & \texttt{avail(soil, gamma)}, \texttt{at(alpha)}\}\end{aligned}$$

$$\begin{aligned}\ell_0^{RP}(\texttt{avail(soil, alpha)}) &= \{s_\alpha\} \\ \ell_0^{RP}(\texttt{avail(soil, beta)}) &= \{s_\beta\} \\ \ell_0^{RP}(\texttt{avail(soil, gamma)}) &= \{s_\gamma\} \\ \ell_0^{RP}(\texttt{at(alpha)}) &= \{s_\alpha, s_\beta, s_\gamma\}\end{aligned}$$

$$\begin{aligned}\mathcal{A}_0^{RP} = \ & \{\texttt{sample(soil, alpha)}, \texttt{drive(alpha, beta)}, \\ & \texttt{drive(alpha, gamma)}, \texttt{avail(soil, beta)}_p, \\ & \texttt{avail(soil, gamma)}_p\}\end{aligned}$$

$$\begin{aligned}\ell_0^{RP}(\texttt{sample(soil, alpha)}) & & &= \{s_\alpha\} \\ \ell_0^{RP}(\texttt{drive(alpha, beta)})) &= \ell_0^{RP}(\texttt{avail(soil, beta)}_p)) & &= \{s_\beta\} \\ \ell_0^{RP}(\texttt{drive(alpha, gamma)}) &= \ell_0^{RP}(\texttt{avail(soil, gamma)}_p)) & &= \{s_\gamma\}\end{aligned}$$

$$\begin{aligned}\mathcal{E}_0^{RP} = \ & \{\varphi_{00}(\texttt{sample(soil, alpha)}), \varphi_{00}(\texttt{drive(alpha, beta)}), \\ & \varphi_{00}(\texttt{drive(alpha, gamma)}), \varphi_{00}(\texttt{avail(soil, beta)}_p), \\ & \varphi_{00}(\texttt{avail(soil, gamma)}_p)\}\end{aligned}$$

$\ell_0^{RP}(\varphi_{00}(\texttt{sample(soil, alpha)})) = \{s_\alpha\}$

$\ell_0^{RP}(\varphi_{00}(\texttt{drive(alpha, beta)})) = \ell_0^{RP}(\varphi_{00}(\texttt{avail(soil, beta)}_p)) = \{s_\beta\}$

$\ell_0^{RP}(\varphi_{00}(\texttt{drive(alpha, gamma)})) = \ell_0^{RP}(\varphi_{00}(\texttt{avail(soil, gamma)}_p)) = \{s_\gamma\}$

$\mathcal{P}_1^{RP} = \{\texttt{have(soil)}, \texttt{avail(soil, beta)},$
$\texttt{at(beta)}, \texttt{avail(soil, gamma)},$
$\texttt{at(gamma)}\}$

$\ell_1^{RP}(\texttt{have(soil)}) = \{s_\alpha\}$

$\ell_1^{RP}(\texttt{avail(soil, beta)}) = \ell_1^{RP}(\texttt{at(beta)}) = \{s_\beta\}$

$\ell_1^{RP}(\texttt{avail(soil, gamma)}) = \ell_1^{RP}(\texttt{at(gamma)}) = \{s_\gamma\}$

$\mathcal{A}_1^{RP} = \{\texttt{commun(soil)}, \texttt{sample(soil, beta)}, \texttt{sample(soil, gamma)}\}$

$\ell_1^{RP}(\texttt{commun(soil)}) = \{s_\alpha\}$

$\ell_1^{RP}(\texttt{sample(soil, beta)}) = \{s_\beta\}$

$\ell_1^{RP}(\texttt{sample(soil, gamma)}) = \{s_\gamma\}$

$\mathcal{E}_1^{RP} = \{\varphi_{00}(\texttt{commun(soil)}), \varphi_{00}(\texttt{sample(soil, beta)}),$
$\varphi_{00}(\texttt{sample(soil, gamma)})\}$

$\ell_1^{RP}(\varphi_{00}(\texttt{commun(soil)})) = \{s_\alpha\}$

$\ell_1^{RP}(\varphi_{00}(\texttt{sample(soil, beta)})) = \{s_\beta\}$

$\ell_1^{RP}(\varphi_{00}(\texttt{sample(soil, gamma)})) = \{s_\gamma\}$

$\mathcal{P}_2^{RP} = \{\texttt{have(soil), comm(soil)}\}$

$\ell_2^{RP}(\texttt{have(soil)}) = \{s_\beta, s_\gamma\}$

$\ell_2^{RP}(\texttt{comm(soil)}) = \{s_\alpha\}$

$\mathcal{A}_2^{RP} = \{\texttt{commun(soil)}, \texttt{comm(soil)}_p\}$

$\ell_2^{RP}(\texttt{commun(soil)}) = \{s_\beta, s_\gamma\}$

$\ell_2^{RP}(\texttt{comm(soil)}_p) = \{s_\alpha\}$

$\mathcal{E}_2^{RP} = \{\varphi_{00}(\texttt{commun(soil)}), \varphi_{00}(\texttt{comm(soil)}_p)\}$

$\ell_2^{RP}(\varphi_{00}(\texttt{commun(soil)})) = \{s_\beta, s_\gamma\}$

$\ell_2^{RP}(\varphi_{00}(\texttt{comm(soil)}_p)) = \{s_\alpha\}$

$\mathcal{P}_3^{RP} = \{\texttt{comm(soil)}\}$

$\ell_3^{RP}(\texttt{comm(soil)}) = \{s_\alpha, s_\beta, s_\gamma\}$

The relaxed plan starts by forming $\mathcal{P}_3^{RP}$ with the goal propositions, namely `comm(soil)`, labeled with each state in b_I because it needs to be supported by all states in the belief state. Next, it supports each proposition in $\mathcal{P}_3^{RP}$ with the relevant effects from $\mathcal{E}_2$ to form $\mathcal{E}_2^{RP}$. The `comm(soil)` proposition is supported first by the persistence of `comm(soil)` from s_α and by the effect of `commun(soil)` in the other two worlds (this is an example of positive interaction of worlds). The `commun(soil)` action is included in $\mathcal{A}_2^{RP}$ for all worlds its effect was *needed*, which is fewer worlds than where it is *reachable*. Next the relaxed plan extraction forms $\mathcal{P}_2^{RP}$ with the execution preconditions of actions in $\mathcal{A}_2^{RP}$ and antecedents of effects in $\mathcal{E}_2^{RP}$, which are `have(soil)` and `comm(soil)`, labeled with all worlds where an action or effect needed it. In the same fashion as level three, the relaxed plan supports the propositions at level two, using `commun(soil)` for state s_α, and one `sample` action for each of the other two states. The relaxed plan extraction continues until all propositions in level one are supported by actions in level zero. To see an example of where an action is used for supporting a proposition in less worlds than it is reachable, consider the `at(beta)` proposition in level one: it is only needed to support `sample(soil, beta)` in world s_β, so does not need to be supported in the other worlds.

For the general case, extraction starts at the level k where the goal propositions are reachable by every state in b (i.e., $b = \bigcap_{p \in G} \ell_k(p)$). The first relaxed plan layers constructed are $\mathcal{A}^{RP}_{k-1}, \mathcal{E}^{RP}_{k-1}, \mathcal{P}^{RP}_k$, where $\mathcal{P}^{RP}_k$ contains all propositions $p \in G$, labeled as $\ell^{RP}_k(p) = b$.

For each level r, $1 \leq r \leq k$, the relaxed plan supports each proposition in $\mathcal{P}^{RP}_{r+1}$ by choosing relevant effects from $\mathcal{E}_r$ to form $\mathcal{E}^{RP}_r$. An effect $\varphi_{ij}(a)$ is relevant if it is reachable in some of the worlds where support for some p is needed (i.e., $\ell_r(\varphi_{ij}(a)) \cap \ell^{RP}_{r+1}(p) \neq \emptyset$) and the consequent gives the proposition p. For each proposition, the relaxed plan must choose enough supporting effects so that the chosen effect worlds are a superset of the worlds we need to support the proposition, formally:

$$\forall_{p \in \mathcal{P}^{RP}_{r+1}} \ell^{RP}_{r+1}(p) \subseteq \left(\bigcup_{\varphi_{ij}(a) \in \mathcal{E}_r : p \in \varepsilon^{+}_{ij}(a)} \ell^{RP}_r(\varphi_{ij}(a)) \right)$$

Supporting a clause in a set of worlds is a set cover problem where effects cover subsets of worlds. The algorithm to cover the worlds of a proposition with worlds of effects is a variant of the well-known greedy algorithm for set cover (Cormen *et al.*, 1990). It chooses all applicable persistence and then action effects that cover the most new worlds, breaking ties by preferring persistence. Each effect chosen for support is added to $\mathcal{E}^{RP}_r$ and labeled with the new worlds it covered for p. Once all propositions in $\mathcal{P}^{RP}_{r+1}$ are covered, the relaxed plan forms the action layer $\mathcal{A}^{RP}_r$ as all actions that have an effect in $\mathcal{E}^{RP}_r$. The actions in $\mathcal{A}^{RP}_r$ are labeled to indicate all worlds where any of their effects were labeled in $\mathcal{E}^{RP}_r$.

The next proposition layer, $\mathcal{P}^{RP}_r$, contains all propositions from the execution preconditions of actions in $\mathcal{A}^{RP}_r$ and antecedents of effects in $\mathcal{E}^{RP}_r$. Each proposition $p \in \mathcal{P}^{RP}_r$ is labeled to indicate all worlds where any action or effect requires p. The relaxed plan supports the propositions in $\mathcal{P}^{RP}_{r-1}$ in the same fashion as $\mathcal{P}^{RP}_r$. The relaxed plan continues to support propositions with effects, insert actions, and insert action and effect preconditions until it has supported all propositions in $\mathcal{P}^{RP}_1$.

The relaxed plan heuristic value, $h^{LUG}_{RP}(b, G)$, is the summation of the number of non persistence actions in each action layer, formally:

$$h_{RP}^{LUG}(b, G) = \sum_{r=0}^{k} \mid \mathcal{A}_r^{RP} \mid$$

Thus in the rover example the heuristic is $h_{RP}^{LUG}(b_I, G) = 7$.

3. Empirical Evaluation: Intra-Planner Comparison

The previous chapter evaluated several different heuristic measures, finding that the h_{RPU}^{MG} (based on measuring state overlap) is most informative despite being very costly to compute. This chapter identifies an alternative way to compute heuristics that measure state overlap based on labeled planning graphs. This section compares the h_{RP}^{LUG} and the h_{RPU}^{MG} relaxed plan heuristics to gauge the effectiveness of the LUG. Specifically, I discuss results for both conformant and conditional planning domains using all possible worlds in a belief state, and also comment on results presented in the previous chapter that compare techniques for state sampling.

Conformant Planning: Table 7 and Figures 28 to 33 compare the two relaxed plan heuristics on all of the conformant planning instances. The LUG heuristic takes considerably less time to compute, leading to better overall time even when it expands more search nodes. The difference between the MG and the LUG are especially pronounced in Cube Center and Ring (Figures 32 and 20), where the size of the initial belief state is quite large as the instances scale. In many domains (e.g., Rovers, CubeCenter and Ring) the LUG heuristic expands significantly more search nodes, because perhaps the relaxed plan extraction makes poor local choices, leading to inferior heuristics. For example, choosing effects that cover the most new worlds may be effectively end-aligning the relaxed plans for different worlds, leading to lower overlap.

Conformant		
Problem	h_{RPU}^{MG}	h_{RP}^{LUG}
Rovers 1	580/6	590/6
2	830/13	680/11
3	1370/23	850/16
4	2170/34	1130/28
5	31480/73	2050/36
6	31950/73	9850/147
Logistics 1	650/9	560/9
2	2310/20	910/15
3	2000/15	1130/14
4	53470/382	3180/46
5	471850/988	6010/42
BT 2	500/2	460/2
10	880/10	520/10
20	6870/20	1230/20
30	44260/30	4080/30
40	183930/40	11680/40
50	758140/50	28420/50
60	TO	59420/60
70	-	113110/70
80	-	202550/80
BTC 2	470/3	460/3
10	990/19	540/19
20	9180/39	1460/39
30	54140/59	4830/59
40	251140/79	14250/79
50	1075250/99	34220/99
60	TO	71650/119
70	-	134880/139
CubeCenter 3	0430/11	70/11
5	14780/82	1780/205
7	1183220/444	27900/1774
9	TO	177790/7226
11	-	609540/17027
13	-	TO
Ring 2	80/8	30/8
3	920/19	70/10
4	19300/40	250/24
5	TO	970/44
6	-	4080/98
7	-	75020/574
8	-	388300/902
9	-	TO
10	-	-

Conditional		
Problem	h_{RPU}^{MG}	h_{RP}^{LUG}
Rovers 1	580/6	580/6
2	730/12	730/13
3	910/17	810/16
4	1070/21	910/21
5	12810/209	7100/174
6	30250/198	13560/174
Logistics 1	600/10	570/10
2	1360/36	1250/36
3	1290/19	1210/19
4	3940/57	4160/57
5	20040/178	20170/178
BT 2	470/3	460/3
10	1020/19	550/19
20	9380/39	1610/39
30	55750/59	5970/59
40	233720/79	17620/79
50	TO	43020/99
60	-	91990/119
70	-	170510/139
80	-	309940/159
BTC 2	470/3	470/3
10	1200/19	590/19
20	11610/39	1960/39
30	64290/59	6910/59
40	274610/79	19830/79
50	TO	49080/99
60	-	103480/119
70	-	202040/139

Table 7. Results for $POND$ for conformant Rovers, Logistics, BT, BTC, Cube Center, and Ring (left), plus results for conditional Rovers, Logistics, BT, BTC (right). The data is Total Time (ms)/# Expanded Nodes, "TO" indicates a time out and "-" indicates no attempt.

Conditional Planning: The conditional planning results in Table 7 and Figures 34 to 37 are a bit more promising. The number of search node expansions is comparable and the time taken using

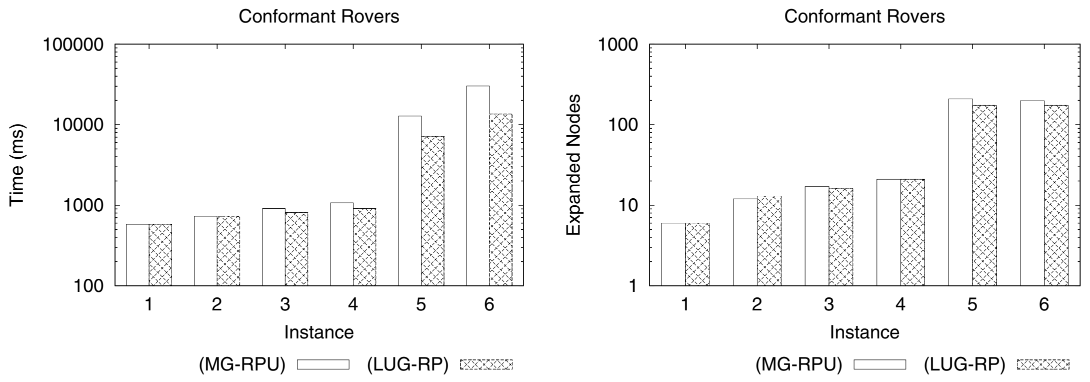


Figure 28. Total Time (ms) on left and Expanded Nodes on right in Conformant Rovers domain.

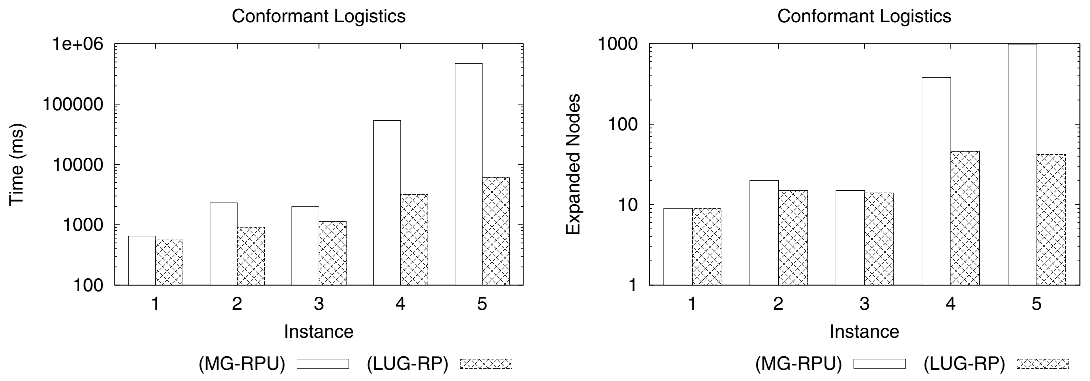


Figure 29. Total Time (ms) on left and Expanded Nodes on right in Conformant Logistics domain.

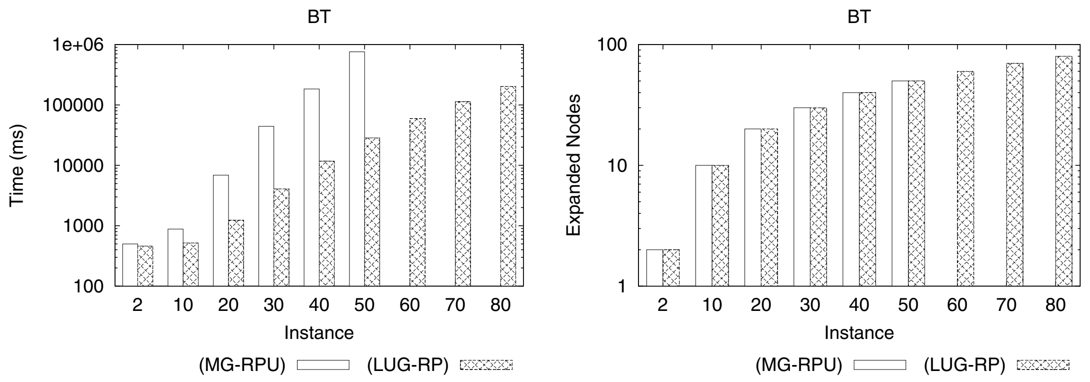


Figure 30. Total Time (ms) on left and Expanded Nodes on right in BT domain.

the LUG heuristic is generally lower. In the BT and BTC domains, scalability is vastly improved, allowing $POND$ to solve more instances.

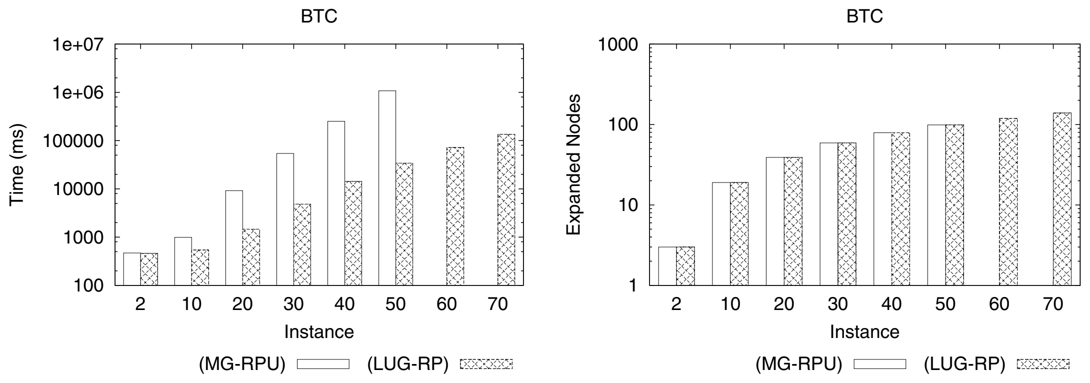


Figure 31. Total Time (ms) on left and Expanded Nodes on right in BTC domain.

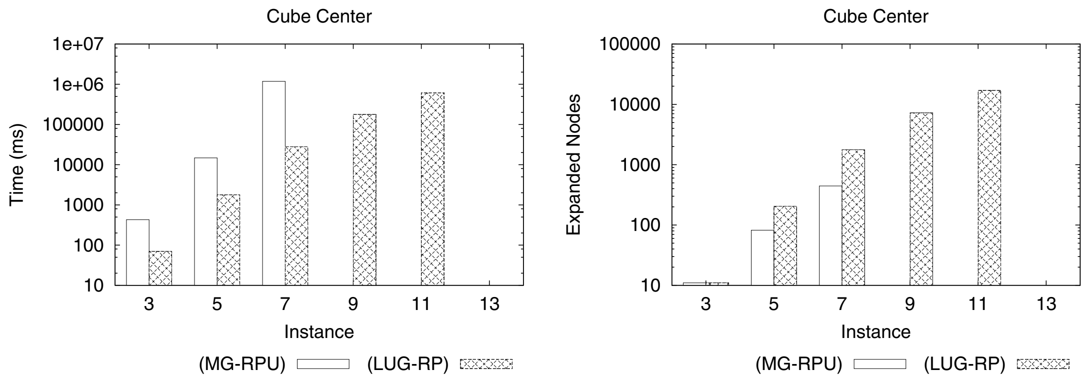


Figure 32. Total Time (ms) on left and Expanded Nodes on right in Cube Center domain.

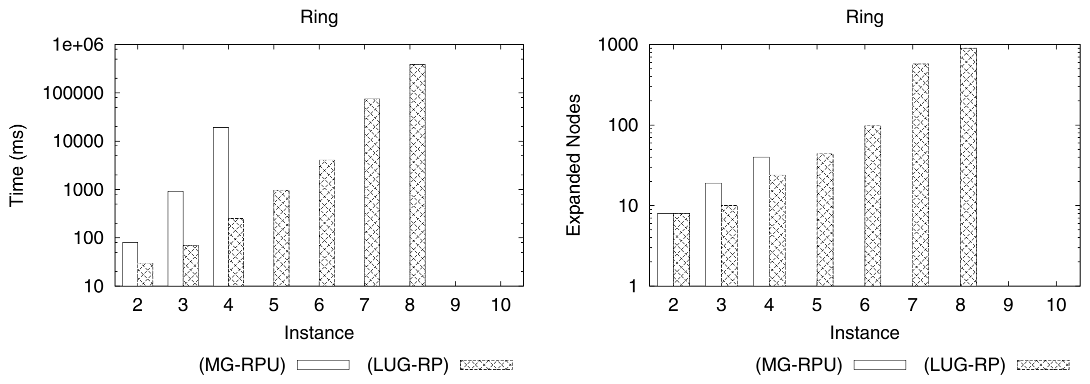


Figure 33. Total Time (ms) on left and Expanded Nodes on right in Ring domain.

Sampling States: The previous chapter presented a study of the effect of sampling states to include in the heuristic computation. Figure 25 showed that using more states was only beneficial in the case of the LUG. This is because the LUG can better accommodate more states with the symbolic

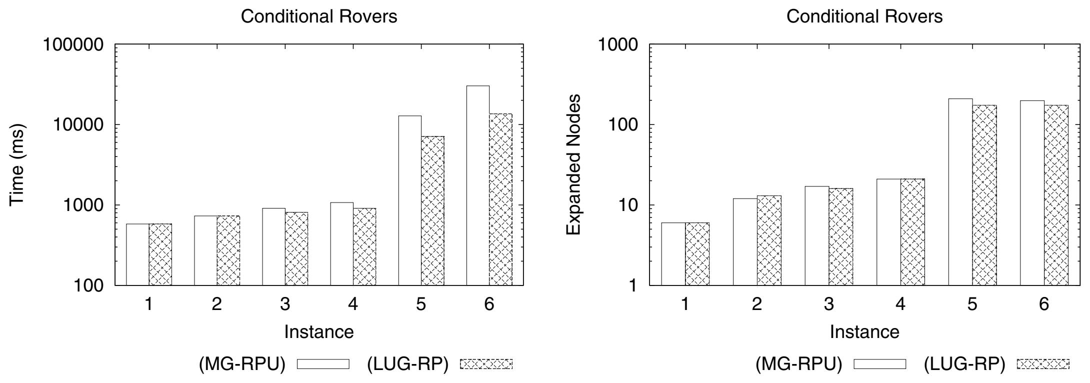


Figure 34. Total Time (ms) on left and Expanded Nodes on right in Conditional Rovers domain.

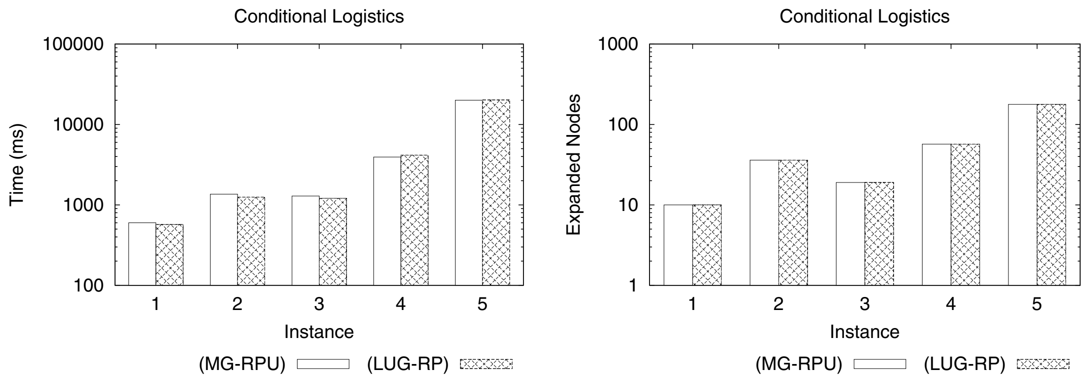


Figure 35. Total Time (ms) on left and Expanded Nodes on right in Conditional Logistics domain.

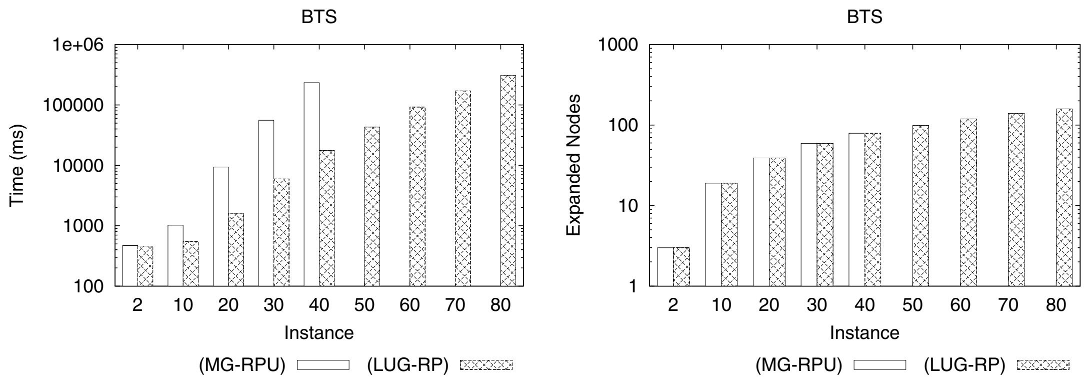
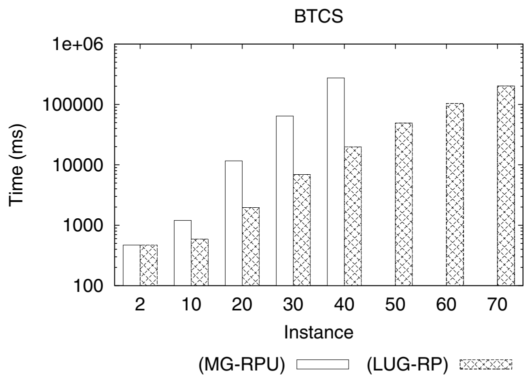


Figure 36. Total Time (ms) on left and Expanded Nodes on right in BTS domain.

representation. The case for MG was much different because despite improved informedness with more states, the cost of constructing the explicit planning graphs is too high.



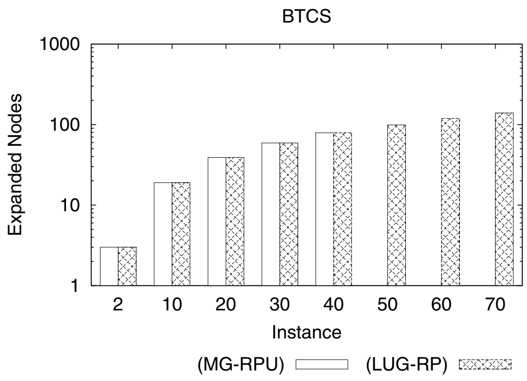


Figure 37. Total Time (ms) on left and Expanded Nodes on right in BTCS domain.

Summary: By comparing graph structures that provide the basis for belief state distance measures, I found that $POND$ is able to exhibit better scalability by using the LUG to optimize the representation of the redundant planning graph structure. The improvement in scalability is attributed to lowering the cost of heuristic computation, but retaining measures of multiple state distances. The LUG makes a trade-off of using an exponential time algorithm for evaluation of labels instead of building an exponential number of planning graphs. This trade-off is justified by our experiments.

4. Empirical Evaluation: Inter-Planner Comparison

This section first compares $POND$ with several conformant planners on the conformant domains, and then compares $POND$ with conditional planners on the conditional domains. I compare with the competing approaches CGP, SGP, GPT v1.40, MBP v0.91, KACMBP, YKA, and CFF. The purpose in this section is to identify the advantages of the planning graph based heuristic techniques over the state of the art planners. I end the section with a discussion of general conclusions drawn from the evaluation.

Planner	Search Space	Search Direction	Conditional	Heuristic	Implementation
$POND$	Belief Space	Forward	$\surd$	Planning Graph	C
MBP	Belief Space	Forward/Backward	$\surd$	Cardinality	C
KACMBP	Belief Space	Forward		Cardinality	C
CGP	Planning Graph	Backward		Planning Graph	Lisp
SGP	Planning Graph	Backward	$\surd$	Planning Graph	Lisp
GPT	Belief Space	Forward	$\surd$	State Space Plans	C
YKA	Belief Space	Backward	$\surd$	Cardinality	C
CFF	Belief Space	Forward		Planning Graph	C

Figure 38. Comparison of planner features.

4.1. Non-Deterministic Conformant Planning. Although this and the previous chapter are aimed at giving a general comparison of heuristics for belief space planning, I also present a comparison of the best heuristic within $POND$ to some of the other leading approaches to conformant planning. Figure 38 lists several features of the evaluated planners, such as their search space, their search direction, whether they are conditional, the type of heuristics, and the implementation language. Note, since each approach uses a different planning representation (BDDs, GraphPlan, etc.), and not all of which even use heuristics, it is hard to get a standardized comparison of heuristic effectiveness. Furthermore, not all of the planners use PDDL-like input syntax; MBP, and KACMBP use AR encodings which may give them an advantage in reducing the number of propositions and actions. I gave the MBP planners the same grounded and filtered action descriptions that I used in $POND$. I also tried, but do not report results, giving the MBP planners the full set of ground actions without filtering irrelevant actions. It appears that the MBP planners do not use any sort of action pre-processing because performance was much worse with the full grounded set of actions. Table 39 and Figures 40 to 45 compare MBP, KACMBP, GPT, CGP, YKA, and CFF with h^{LUG}_{RP} in $POND$ with respect to run time and plan length.

MBP: The MBP planner uses a cardinality heuristic that in many cases overestimates plan distances (as per the discussion of h_{card} in the last chapter). MBP uses regression search for conformant plans, but progression search for conditional plans. It is interesting to note that in the more difficult

Problem	POND h_{RP}^{LUG}	MBP	KACMBP	GPT	CGP	YKA	CFF
Rovers 1	590/5	66/5	9293/5	3139/5	70/5	1220/7	70/5
2	680/9	141/8	9289/15	4365/8	180/8	2050/10	30/8
3	850/11	484/10	9293/16	5842/10	460/10	1740/12	10/10
4	1130/16	3252/13	9371/18	7393/13	1860/13	2010/16	10/13
5	2050/25	OoM	39773/40	399525/20	OoM	7490/27	18/22
6	8370/25	727/32	TO	TO	-	24370/26	21/23
Logistics 1	560/9	37/9	127/12	916/9	60/6	250/13	10/9
2	910/15	486/24	451/19	1297/15	290/6	670/19	12/15
3	1130/14	408/14	1578/18	1711/11	400/8	20280/21	14/12
4	3180/22	2881/27	8865/22	9828/18	1170/8	17530/27	12/18
5	6010/29	OoM	226986/42	543865/28	TO	141910/40	25/28
BT 2	460/2	6/2	10/2	487/2	20/1	0/2	0/2
10	520/10	119/10	16/10	627/10	520/1	0/10	30/10
20	1230/20	80/20	84/20	472174/20	3200/1	20/20	4400/20
30	4080/30	170/30	244/30	TO	10330/1	80/30	4500/30
40	11680/40	160/40	533/40	-	24630/1	160/40	26120/40
50	28420/50	300/50	1090/50	-	49329/1	250/50	84730/50
60	59420/60	480/60	2123/60	-	87970/1	420/60	233410/60
70	113110/70	730/70	3529/70	-	145270/1	620/70	522120/70
80	202550/80	1080/80	1090/80	-	TO	3310/80	979400/80
BTC 2	460/3	8/3	18/3	465/3	0/3	10/3	10/3
10	540/19	504/19	45/19	715/19	39370/19	30/19	57/19
20	1460/39	98/39	211/39	-	-	240/39	2039/39
30	4820/59	268/59	635/59	-	-	1210/59	23629/59
40	14250/79	615/79	1498/79	-	-	3410/79	116156/79
50	34220/99	1287/99	10821/99	-	-	8060/50	334879/99
60	71650/119	2223/119	5506/119	-	-	15370/119	TO
70	134880/139	3625/139	2640/139	-	-	27400/139	-
CubeCenter 3	70/9	10/9	20/9	40/9	28990/3	0/9	20/15
5	1780/18	16/18	20/18	363/18	TO	0/19	28540/45
7	27900/29	35/27	70/27	4782/27	-	20/34	TO
9	177790/36	64/36	120/36	42258/36	-	80/69	-
11	609540/47	130/45	230/45	26549/45	-	190/68	-
Ring 2	30/6	0/5	0/5	31/5	TO	0/5	360/12
3	70/8	0/8	40/8	35/8	-	0/8	TO
4	250/13	10/11	30/11	60/11	-	20/11	-
5	970/17	20/14	50/14	635/14	-	80/14	-
6	4080/22	30/17	120/18	51678/17	-	110/17	-
7	75020/30	80/20	230/21	TO	-	300/20	-
8	388300/29	160/23	600/24	-	-	480/23	-

Figure 39. Results for $POND$ using h_{RP}^{LUG} compared with MBP, KACMBP, GPT, CGP, YKA, and CFF for conformant Rovers, Logistics, BT, BTC, Cube Center, and Ring. The data is Total Time / # Plan Steps, "TO" indicates a time out (20 minutes), "OoM" indicates out of memory (1GB), and "-" indicates no attempt.

problem instances in the Rovers and Logistics domains MBP and KACMBP tend to generate much longer plans than the other planners. MBP does outperform $POND$ in some cases but does not find solutions in certain instances (like Rovers 5), most likely because of its heuristic. Note that KACMBP and MBP are quite fast on the Cube Center and Ring domains, but have more trouble on domains like Rovers and Logistics. This illustrates how a heuristic modeling knowledge as opposed to reachability can do well in domains where the challenge is uncertainty not reachability.

Optimal Planners: The optimal approaches (CGP and GPT) tend not to scale as well, despite their good solutions. CGP has trouble constructing its planning graphs as the parallel conformant plan depth increases. CGP spends quite a bit of time computing mutexes, which increases the planning cost as plan lengths increase. CGP does much better on shallow and parallel domains like BT, where it can find one step plans that dunk every package in parallel. GPT performs progression search that is guided by a heuristic that measures the cost of fully observable plans in state space. GPT finds optimal serial plans but is not as effective when the size of the search space increases. GPT fails to scale with the search space because it becomes difficult to even compute its heuristic (due to a larger state space as well).

YKA: YKA is a regression planner that uses a BDD based representation and a cardinality heuristic. YKA performs well on all the domains without reachability heuristics because of its search engine based on BDDs. $POND$ is able to do better than YKA in the Rovers and Logistics domains, but it is unclear whether this is because of the search direction or heuristics.

CFF: Conformant FF, a progression planner using a relaxed plan similar to the LUG relaxed plan, does very well in the Rovers and Logistics domains because it uses the highly optimized FF search engine as well as a cheap to compute relaxed plan heuristic. However, CFF does not do as well in the BT, BTC, Cube Center, and Ring problems because there are not as many propositions that will be entailed by a belief state. CFF relies on implicitly representing belief states in terms of the

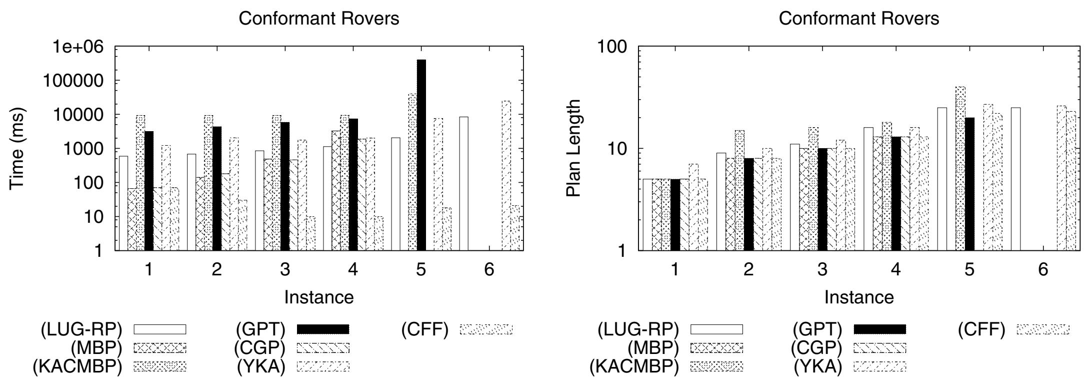


Figure 40. Total Time (ms) on left and Plan Length on right in Conformant Rovers domain.

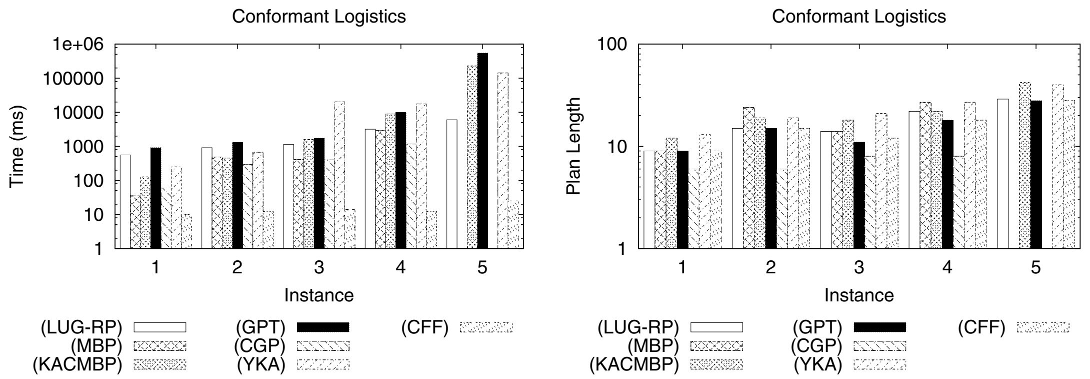


Figure 41. Total Time (ms) on left and Plan Length on right in Conformant Logistics domain.

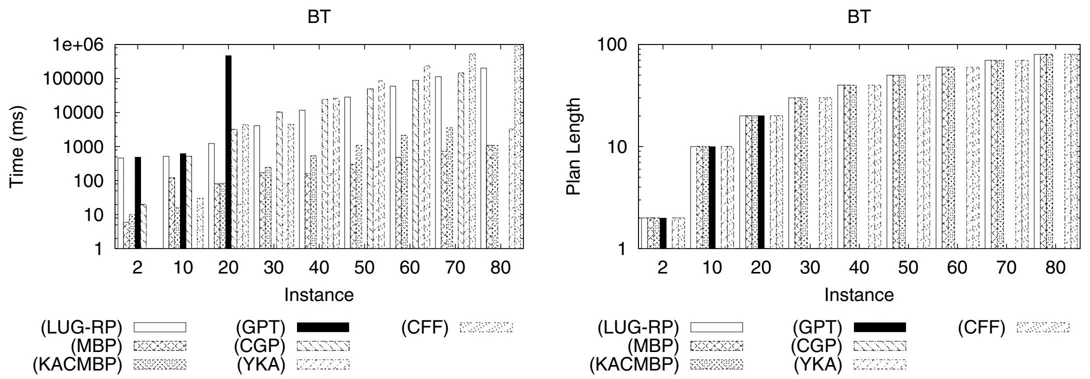


Figure 42. Total Time (ms) on left and Plan Length on right in BT domain.

literals that are entailed by the belief state, the initial belief state, and the action history. When there are very few literals that can be entailed by the belief state, reasoning about the belief state requires inference about the action history. Another possible reason CFF suffers is the problem encodings.

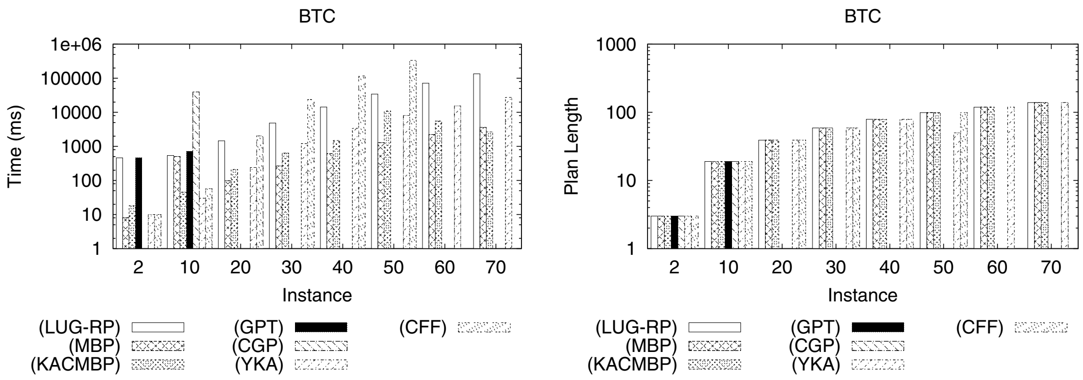


Figure 43. Total Time (ms) on left and Plan Length on right in BTC domain.

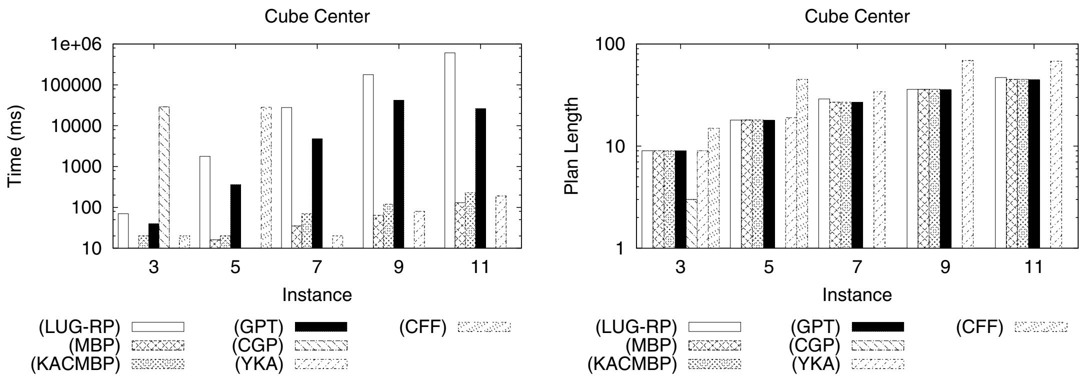


Figure 44. Total Time (ms) on left and Plan Length on right in Cube Center domain.

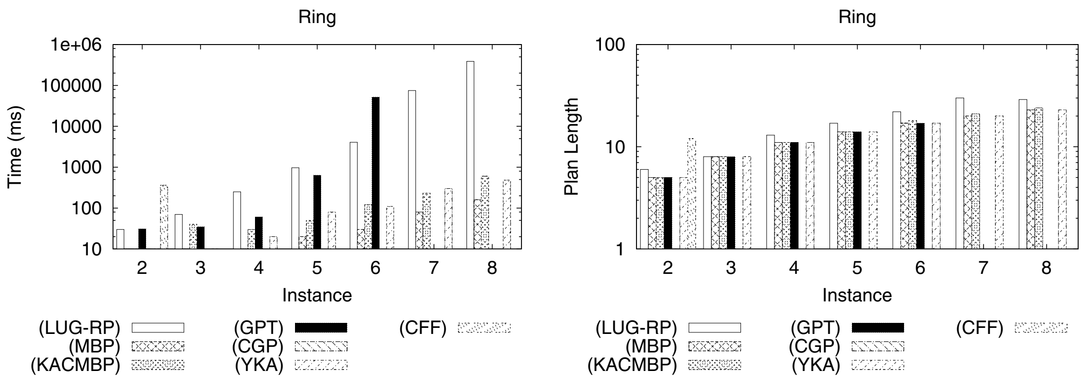


Figure 45. Total Time (ms) on left and Plan Length on right in Ring domain.

The Cube Center and Ring domains are naturally expressed with multi-valued state features, and in our transformation to binary state features I describe the values that must hold but also the values that must not hold. This is difficult for CFF because the conditional effect antecedents contain

Problem	POND h_{RP}^{LUG}	MBP	GPT	SGP	YKA
Rovers 1	580/5	3312/11	3148/5	70/5	3210/5
2	730/8	4713/75	5334/7	760/7	6400/7
3	810/8	5500/119	7434/8	TO	7490/8
4	910/10	5674/146	11430/10	-	11210/10
5	7100/19	16301/76	TO	-	TO
6	13560/22	OoM	-	-	-
Logistics 1	570/7	41/16	1023/7	5490/6	1390/8
2	1250/12	22660/177	5348/12	TO	TO
3	1210/9	2120/45	2010/8	-	TO
4	4160/15	OoM	TO	-	-
5	20170/22	-	-	-	-
BT 2	460/2	0/2	510/2	0/1	0/2
10	550/10	240/10	155314/10	70/1	20/10
20	1610/20	OoM	OoM	950/1	60/20
30	5970/30	-	-	4470/1	200/30
40	17620/40	-	-	13420/1	400/40
50	43020/50	-	-	32160/1	810/50
60	91990/60	-	-	90407/1	1350/60
70	170510/70	-	-	120010/1	2210/70
80	309940/80	-	-	TO	3290/80
BTC 2	470/2	20/2	529/2	10/2	0/4
10	590/10	280/10	213277/10	TO	210/12
20	1960/20	OoM	TO	-	2540/22
30	6910/30	-	-	-	13880/32
40	19830/40	-	-	-	46160/42
50	49080/50	-	-	-	109620/52
60	103480/60	-	-	-	221460/62
70	202040/70	-	-	-	41374/72

Figure 46. Results for $POND$ using h_{RP}^{LUG} compared with MBP, GPT, SGP, and YKA for conditional Rovers, Logistics, BT, and BTC. The data is Total Time / # Maximum possible steps in a execution, "TO" indicates a time out (20 minutes), "OoM" indicates out of memory (1GB), and "-" indicates no attempt.

several propositions and its heuristic is restricted to considering only one such proposition. It may be that CFF is choosing the wrong proposition or simply not enough propositions to get effective heuristics. However in BT and BTC, where I used only one proposition in effect antecedents CFF still performs poorly.

4.2. Non-Deterministic Conditional Planning. Table 46 and Figures 47 to 50 show the results for testing the conditional versions of the domains on $POND$, MBP, GPT, SGP, and YKA.

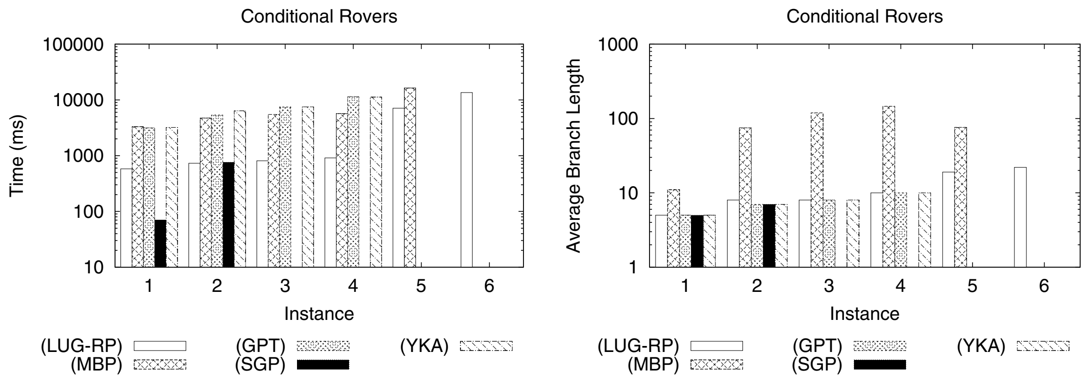


Figure 47. Total Time (ms) on left and Average Branch Length on right in Conditional Rovers domain.

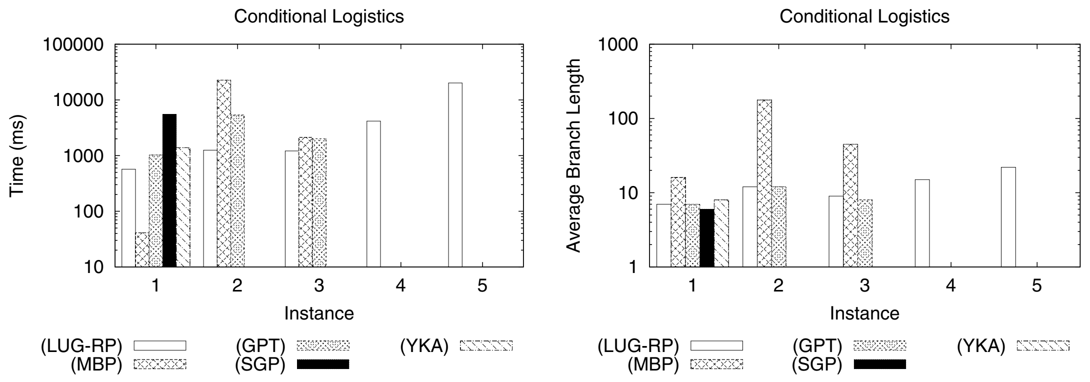


Figure 48. Total Time (ms) on left and Average Branch Length on right in Conditional Logistics domain.

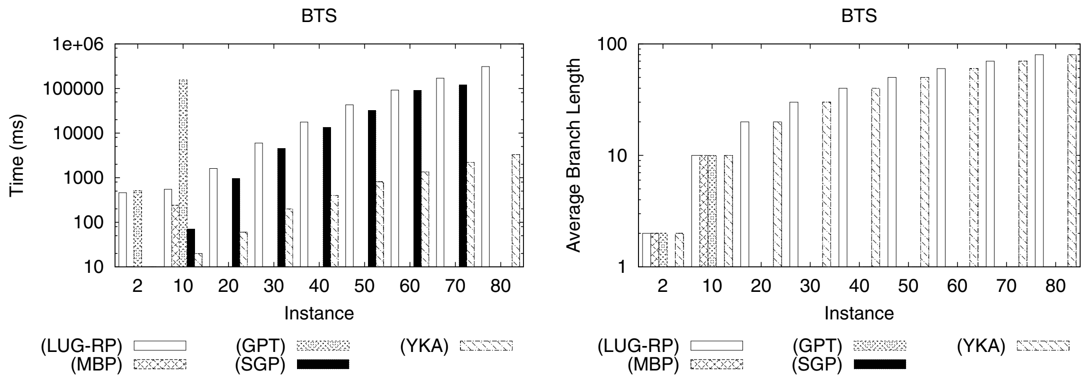
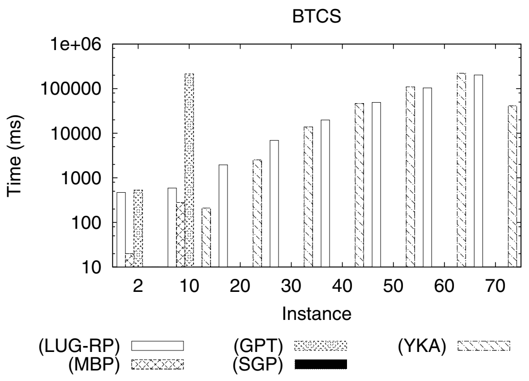


Figure 49. Total Time (ms) on left and Average Branch Length on right in BTS domain.

MBP: The $POND$ planner is very similar to MBP in that it uses progression search. However, $POND$ uses an AO* search, whereas the MBP binary uses a depth first And-Or search. The depth



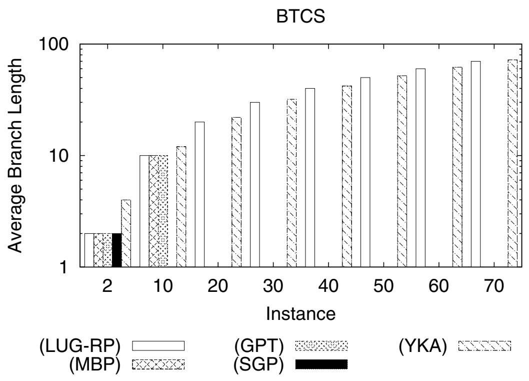


Figure 50. Total Time (ms) on left and Average Branch Length on right in BTCS domain.

first search used by MBP contributes to highly sub-optimal maximum length branches (as much as an order of magnitude longer than $POND$). For instance, the plans generated by MBP for the Rovers domain have the rover navigating back and forth between locations several times before doing anything useful; this is not a situation beneficial for actual mission use. MBP tends to not scale as well as $POND$ in all of the domains tested. A possible reason for the performance of MBP is that the Logistics and Rovers domains have sensory actions with execution preconditions, which prevent branching early and finding deterministic plan segments for each branch. I experimented with MBP using sensory actions without execution preconditions and it was able to scale somewhat better, but plan quality was much longer.

Optimal Planners: GPT and SGP generate better solutions but very slowly. GPT does better on the Rovers and Logistics problems because it exhibits some positive interaction in the plans, but SGP does well on BT because its planning graph search is well suited for shallow, yet broad (highly parallel) problems.

YKA: YKA fares similar to GPT in Rovers and Logistics, but has trouble scaling for other reasons. I think that YKA may be having trouble in regression because of sensory actions since it was able to scale reasonably well in the conformant version of the domains. Despite this, YKA proves to do very well in the BT and BTC problems.

4.3. Empirical Evaluation Conclusions. In the internal comparisons of heuristics within $POND$, as well as external comparisons with several state of the art conformant and conditional planners, this investigation has uncovered many interesting lessons about heuristics for planning in belief space.

- Distance based heuristics for belief space search help control conformant and conditional plan length because, as opposed to cardinality, the heuristics model desirable plan quality metrics.
- Planning graph heuristics for belief space search scale better than planning graph search and admissible heuristic search techniques.
- The LUG is an effective planning graph data structure for search heuristics in belief space.
- Sampling possible worlds to construct planning graphs does reduce computational cost, but considering more worlds by exploiting planning graph structure common to possible worlds (as in the LUG), can be more efficient and informed.
- The LUG heuristics help the conditional planner, $POND$, to scale up in conditional domains, despite the fact that the heuristic computation does not model observation actions.

5. Related Work

The way that the *LUG* handles uncertainty through labels and label propagation is reminiscent of and related to de Kleer's assumption based truth maintenance system (ATMS) (de Kleer, 1986). Where an ATMS uses labels to identify the assumptions (contexts) where a particular statement holds, a traditional truth maintenance system requires extensive backtracking and consistency enforcement to identify other contexts. Similarly, where we can reason about multiple possible worlds (contexts) with the LUG simultaneously, the MG approach requires, not backtracking, but reproduction of planning graphs for other possible worlds.

More recently, there has been closely related work on heuristics for constructing conformant plans within the CFF planner (Brafman and Hoffmann, 2004). The planner represents belief states implicitly through a set of known facts, the action history (leading to the belief state), and the initial belief state. CFF builds a planning graph forward from the set of known literals to the goal literals and backwards to the initial belief state. In the planning graph, conditional effects are restricted to single literals in their antecedent to enable tractable 2-cnf reasoning. From this planning graph, CFF extracts a relaxed plan that represents supporting the goal belief state from all states in the initial belief state. The biggest differences between the LUG and the CFF technique are that the LUG reasons only forward from the source belief state (assuming an explicit, albeit symbolic, belief state), and the LUG does not restrict the number of literals in antecedents. As a result, the LUG does not lose the causal information nor perform backward reasoning to the initial belief state.

The *LUG* is also closely related to work on uncertain databases with lineage (Benjelloun *et al.*, 2006). Uncertain databases can capture incompleteness in element attributes similar to how belief states capture uncertainty about propositions. To process queries, such as joins, on uncertain databases it is important to track the lineage intermediate query products. Tracking lineage is similar to how I propagate labels through the planning graph.

6. Conclusion

In addition to the contributions of the previous chapter, I have:

- Found that exploiting planning graph structure to reduce the cost of considering more possible states of a belief state in heuristics is preferable to sampling a subset of the states for the heuristics.

- Shown that a labeled uncertainty graph can capture the same support information as multiple graphs, and reduces the cost of heuristic computation.
- Shown that the labeled uncertainty graph is very useful for conformant planning and, without considering observational actions and knowledge, can perform well in contingent planning.

I have shown that planning with a Labeled Uncertainty planning Graph LUG, a condensed version of the multiple graphs, is useful for encoding conformant reachability information. The main innovation is the idea of "labels" – labels are attached to all propositions, actions, and effect relations to indicate the set of worlds in which those respective vertices hold. The experimental results show that the LUG can outperform the multiple graph approach. In comparison to other approaches, I have also been able to demonstrate the utility of structured reachability heuristics in controlling plan length and boosting scalability for both conformant and contingent planning.

In chapter 7, I extend the LUG within the framework of state agnostic planning graphs (Cushing and Bryce, 2005). A state agnostic planning graph is essentially a multiple source planning graph, where by analogy a conventional planning graph has a single source. Planning graphs are already multiple destination, so in our generalization the state agnostic planning graph allows us to compute the heuristic distance measure between any pair of states or belief states. The LUG seeks to avoid redundancy across the multiple planning graphs built for states in the same belief state. I extended this notion to avoid redundancy in planning graphs built for every belief state. I will show that the state agnostic LUG ($SLUG$) which is built once per search episode (as opposed to a LUG at each node) can reduce heuristic computation cost without sacrificing informedness.

CHAPTER 5

LABELED PLANNING GRAPHS FOR COST-SENSITIVE PLANNING

While POMDPs provide a general platform for non-deterministic conditional planning under a variety of quality metrics they have limited scalability. On the other hand, non-deterministic conditional planners scale very well, but many lack the ability to optimize plan quality metrics. This chapter presents a novel generalization of planning graph based heuristics that helps conditional planners both scale and generate high quality plans when using actions with non-uniform costs. The technique propagates cost of the *LUG* to extract cost-sensitive relaxed plans. I make empirical comparisons with two state of the art planners to show the benefit of the techniques. This chapter focusses on non-deterministic conformant and conditional planning with deterministic actions.

1. Introduction

In this chapter, I describe a way of extending the reachability heuristics based on the LUG to make them more sensitive to cost/quality information. Classical planning reachability heuristics incorporate cost information by propagating estimates of the cost to reach propositions in the planning graph at different levels. Relaxed plans can then use the cost information to select the least costly supporting action for each proposition. Recall that chapter 2 describes level-based reachability heuristics that define the cost to reach propositions in terms of the first level they appear. When actions have non-uniform costs, this assumption can fail because it may take only level for a proposition to appear, but it could be supported by a high cost action. It is possible that after two levels the proposition could be supported by a different sequence of actions whose total cost is lower. By propagating costs over the planning graph, it is possible to capture the least cost to support propositions at different levels. When given the choice to support a proposition, it should be at the level with lowest cost via the lowest cost actions.

Straightforward adaptation of cost propagation to the LUG unfortunately proves to be infeasible. Recall that the *LUG* relaxed plan may support actions and propositions in different subsets of possible worlds. By propagating costs for different subsets of the possible worlds, it is possible to reason about the cost of supporting propositions in different subsets of possible worlds. On one extreme, each proposition and action can be given a cost for each subset of possible worlds. On the other extreme, cost propagation in the LUG can ignore the possible worlds and assign a single cost to each proposition and action. The former requires an doubly-exponential number of costs and the latter assumes full positive interaction between possible worlds (which may lead to bad heuristics). Even the intermediate approach of tracking a single cost per possible world still requires an exponential number of costs. Instead, I describe a technique to use a small finite number of costs per proposition to approximate the costs of supporting them in different sets of possible worlds.

I describe this method for propagating cost information over the LUG in an extension called the $CLUG$. The $CLUG$ tracks cost over each element of a partition of world sets for each proposition. The size of the partition (number of costs tracked) is bounded by the number of planning graph levels. Each disjoint set is the worlds in which a proposition or action is newly reached at a level. The $CLUG$ is then used as the basis for doing reachability analysis. In particular, I extract relaxed plans from it (as described in the previous chapter), using the cost information to select low cost relaxed plans. The results show that cost-sensitive heuristics improve plan quality and scalability.

I proceed by describing the $CLUG$, and an associated relaxed plan extraction procedure. I present an empirical study of the techniques within $POND$ and compare with two state of the art conditional planners MBP (Bertoli *et al.*, 2001) and GPT (Bonet and Geffner, 2000). I end by providing a comparison to related work, and a conclusion.

2. Cost Labeled Uncertainty Graph ($CLUG$)

In this section, I present the extension of the LUG used to handle costs, called the $CLUG$. Recall that the LUG is a single planning graph that uses an annotation on vertices (actions, effects, and propositions) to reflect assumptions about how the vertex is reached. Specifically it uses a label, $\ell_k(p)$, to denote the states of the current (source) belief b that reach the proposition in level k. The $CLUG$ additionally uses a cost vector $c_k(p)$ to estimate of the cost of reaching the vertex from different states of the source belief.

Consider an extension to the non-deterministic conformant rovers problem to include actions with non-uniform costs, as:

Action	Cost
`drive(alpha, beta)`	10
`drive(beta, alpha)`	5
`drive(beta, gamma)`	15
`drive(gamma, beta)`	10
`drive(alpha, gamma)`	30
`drive(gamma, alpha)`	5
`sample(soil, alpha)`	20
`sample(soil, beta)`	20
`sample(soil, beta)`	20
`commun(soil)`	5

Driving between locations incurs different costs, due to slope and soil consistency. The initial belief state and goal are identical to the formulation previously described for the conformant rovers problem.

Figure 51 illustrates the $CLUG$ built for the initial belief in the example. Label propagation

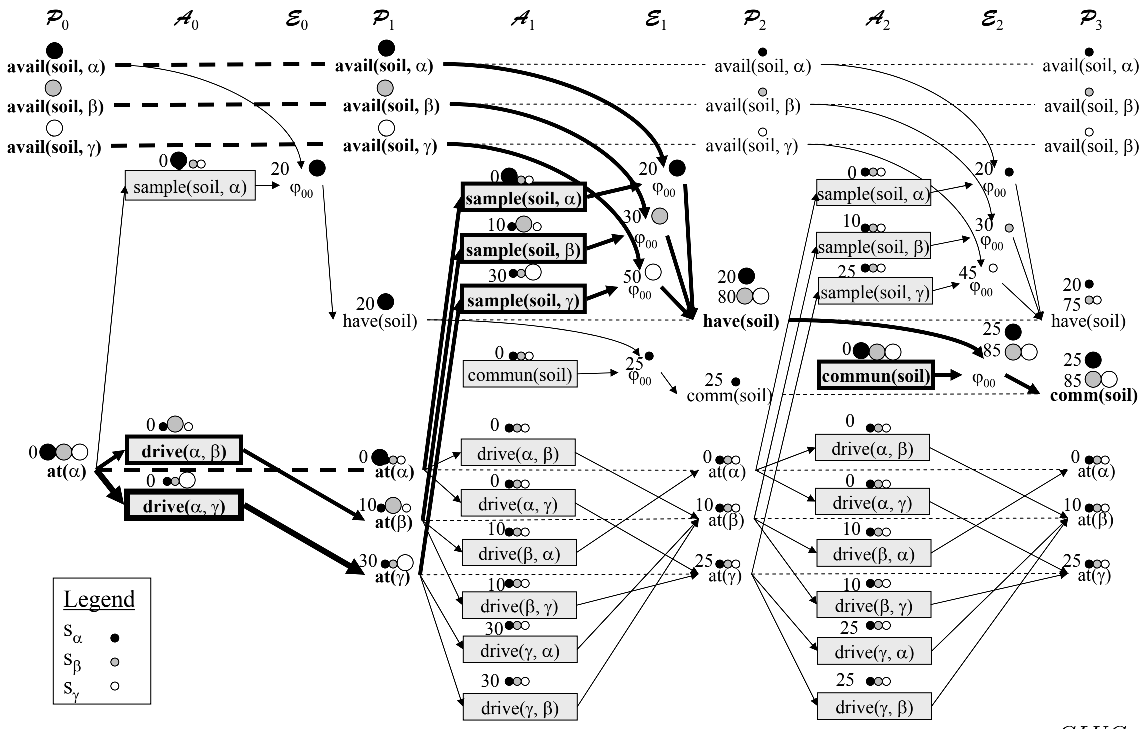


Figure 51. The $CLUG$ for b_I of the conformant rover problem. The relaxed plan for h_{RP}^{CLUG} is shown in bold.

is exactly as described for the LUG in the previous chapter. In the $CLUG$, the labels are partitioned and a cost is assigned to each disjoint subset. In the figure some partitions contain only one set. Some partitions do not have costs depicted because the cost is zero.

Cost propagation on planning graphs, similar to that used in the Sapa planner (Do and Kambhampati, 2003), computes the estimated cost of reaching propositions at different levels, which may decrease over the levels. Since the $CLUG$ tracks whether a proposition is reached in more than one possible world, it is possible that the cost of reaching a proposition is different for every subset of these worlds. Instead of tracking costs for an exponential number of subsets, or even each individual world, it partitions labels into fixed sets to track cost over (i.e., the elements of the cost vectors $c_k(\cdot)$). A cost vector $c_k(\cdot)$ is a partition of worlds represented by the label $\ell_k(\cdot)$ that assigns a cost to each of the disjoint sets.

Definition 9 (Cost Vectors)**.** *A cost vector $c_k(\cdot)$ is a set of pairs $(\ell_k^r(\cdot), c_k^r(\cdot))$, where $\ell_k^r(\cdot) \subseteq \ell_k(\cdot)$ is a set of states and $c_k^r(\cdot)$ is a rational number, such that $\cup_r \ell_k^r(\cdot) = \ell_k(\cdot)$. Every $c_k^r(\cdot)$ is an estimate of the cost of jointly reaching an action, effect, or proposition from all states $s \in \ell_k^r(\cdot)$.*

As I will show, the partitions are different for each action, effect, and proposition because the partition is with respect to the new worlds that reach the given action, effect, or proposition in each level. The reason for defining the partitions this way is that the size of the partition is bounded by the number of $CLUG$ levels.

The $CLUG$ construction requires first defining costs for the initial proposition layer, and then an inductive step to assign costs within each graph level. For each graph layer of the $CLUG$, I compute the label of each vertex $\ell_k(\cdot)$ and update the cost vector of the vertex.

Initial Proposition Layer: The cost vector for each proposition $p \in \mathcal{P}_0$ in the initial layer of the $CLUG$ is defined as $c_0(p) = \{(\ell_0(p), 0)\}$. In the cost vector, I store a cost of zero for the entire group of states in which each proposition is initially reachable.

The cost vectors for the initial proposition layer of the rover problem are as follows:

$$\begin{aligned}
c_0(\texttt{avail(soil, alpha)}) &= \{(\{s_\alpha\}, 0)\} \\
c_0(\texttt{avail(soil, beta)}) &= \{(\{s_\beta\}, 0)\} \\
c_0(\texttt{avail(soil, gamma)}) &= \{(\{s_\gamma\}, 0)\} \\
c_0(\texttt{at(alpha)}) &= \{(\{s_\alpha, s_\beta, s_\gamma\}, 0)\}
\end{aligned}$$

Action Layer: The action layer $\mathcal{A}_k$ of the LUG assigns the cost vector to each action in the layer as: $c_k(a) = \{(\ell_k^r(a), c_k^r(a)) | \ell_k^r(a) \neq \emptyset\}$, each cost vector partition as: $\ell_k^r(a) = \ell_{k'}(a) \backslash \bigcup_{k' \leq k} \ell_{k'-1}(a)$, and each cost as: $c_k^r(a) = \sum\limits_{p \in \rho_e(a)} \texttt{Cover}(\ell_k^r(a), c_k(p))$.

The $\texttt{Cover}(\ell, c)$ operation, detailed below, computes the cost of supporting states in the set ℓ with the cost function c. The action layer $\mathcal{A}_k$ partitions the cost vector of each action based on worlds that newly support the action in each level. If there are new worlds supporting a at level k, the $CLUG$ adds a cost for the new worlds to the cost vector with the label equal to $\ell_k(a) \backslash \ell_{k-1}(a)$. When $k = 0$, assume $\ell_{-1}(a) = \emptyset$. Then the cost for each element of the cost vector is updated. The updating finds $c^r_k(a)$ by summing the costs of the covers of the execution precondition propositions in the worlds described by $\ell^r_k(a)$. The cost of each proposition is determined by covering the worlds $\ell^r_k(a)$ with the cost vector of the proposition. In general, cost vectors do not have a specific pair for the set of worlds under consideration, rather the worlds are partitioned over several pairs. To get a cost for the set of worlds under consideration, the $CLUG$ must cover the worlds with the disjoint world sets in the cost vector. It is best to find a minimum cost for the cover because planning graphs typically represent an optimistic projection of reachability.

Cover(ℓ, c)**:** A `Cover` of a set of worlds ℓ with a set of pairs $c = \{(\ell^1(\cdot), c^1(\cdot)), ..., \ell^n(\cdot), c^n(\cdot))\}$, is equivalent to a weighted set cover problem (Cormen *et al.*, 1990) where the states in ℓ must be covered with weighted sets of states defined by the pairs in c. A set of pairs $c' \subseteq c$ covers ℓ with cost $\sum_{i:(\ell^i(\cdot),c^i(\cdot))\in c'} c^i(\cdot)$ when $\ell \subseteq \bigcup_{i:(\ell^i(\cdot),c^i(\cdot))\in c'} \ell^i(\cdot)$. Finding a minimum cover is an NP-Complete problem, following from set cover. I solve it using a greedy algorithm that at each step chooses the least cost formula-cost pair that covers an uncovered world of ℓ. Fortunately in the action and effect layers, the Cover operation is done with (non-overlapping) partitions, meaning there is only one possible cover. This is not the case in the proposition layer construction and relaxed plan extraction because the cover is with a set of possibly overlapping sets. I show an example of using Cover after the proposition layer definition.

The cost vectors for the zeroth action layer are as follows:

$$
\begin{array}{lcl}
c_0(\texttt{sample(soil, alpha)}) = c_0(\texttt{drive(alpha, beta)}) & = & \\
c_0(\texttt{drive(alpha, gamma)}) = c_0(\texttt{at(alpha)}_p) & = & \{(\{s_\alpha, s_\beta, s_\gamma\}, 0)\} \\
c_0(\texttt{avail(soil, alpha)}_p) & = & \{(\{s_\alpha\}, 0)\} \\
c_0(\texttt{avail(soil, beta)}_p) & = & \{(\{s_\beta\}, 0)\} \\
c_0(\texttt{avail(soil, gamma)}_p) & = & \{(\{s_\gamma\}, 0)\}
\end{array}
$$

Each cost vector has a single element because every possible world labeling the actions is a new world at this level. The costs associated with each element of each cost vector are defined in terms of the precondition propositions, which are each zero. Covering the action cost vector elements requires a single precondition cost vector element in each case.

Effect Layer: In the effect layer $\mathcal{E}_k$, the cost vector of each effect $\varphi_{ij}(a) \in \mathcal{E}_k$ is defined as: $c_k(\varphi_{ij}(a)) = \{(\ell^r_k(\varphi_{ij}(a)), c^r_k(\varphi_{ij}(a)))|\ell^r_k(a) \neq \perp\}$, each cost vector partition of the label as: $\ell^r_k(\varphi_{ij}(a)) = \ell_{k'}(\varphi_{ij}(a)) \backslash \ell_{k'-1}(\varphi_{ij}(a)), k' \leq k$, and each partition subset cost as: $c^r_k(\varphi_{ij}(a)) = c(a) + \texttt{Cover}(\ell^r_k(\varphi_{ij}(a)), c_k(a)) + \sum_{p \in \rho_{ij}(a)} \texttt{Cover}(\ell^r_k(\varphi_{ij}(a)), c_k(p))$. The cost $c^r_k(a)$ of state set $\ell^r_k(a)$ 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 $\ell^r_k(\varphi_{ij}(a))$, and the support cost of the antecedent in $\ell^r_k(\varphi_{ij}(a))$ (found by summing over the cost of each proposition $p \in \rho_{ij}(a)$).

The zeroth effect layer for our example has the cost vectors:

$$
\begin{array}{lcl}
c_0(\varphi_{00}(\texttt{sample(soil, alpha)})) & = & \{(\{s_\alpha\}, 20)\} \\
c_0(\varphi_{00}(\texttt{drive(alpha, beta)})) & = & \{(\{s_\alpha, s_\beta, s_\gamma\}, 10)\} \\
c_0(\varphi_{00}(\texttt{drive(alpha, gamma)})) & = & \{(\{s_\alpha, s_\beta, s_\gamma\}, 30)\} \\
c_0(\varphi_{00}(\texttt{at(alpha)}_p)) & = & \{(\{s_\alpha, s_\beta, s_\gamma\}, 0)\} \\
c_0(\varphi_{00}(\texttt{avail(soil, alpha)}_p)) & = & \{(\{s_\alpha\}, 0)\} \\
c_0(\varphi_{00}(\texttt{avail(soil, beta)}_p)) & = & \{(\{s_\beta\}, 0)\} \\
c_0(\varphi_{00}(\texttt{avail(soil, gamma)}_p)) & = & \{(\{s_\gamma\}, 0)\}
\end{array}
$$

Like the action layer, each effect cost vector contains a single element because the set of worlds labeling each effect is new at this level. A single element of each precondition and associated action is needed to cover the each element of the effect cost vectors.

Proposition Layer: The propositions layer $\mathcal{P}_k$ defines each cost vector of each proposition $p \in \mathcal{P}_k$ as: $c_k(p) = \{(\ell_k^r(p), c_k^r(p)) | \ell_k^r(a) \neq \emptyset\}$, each partition as: $\ell_k^r(a) = \ell_{k'}(p) \setminus \bigcup_{k' \leq k} \ell_{k'-1}(p)$, and each subset's cost as: $c_k^r(a) = \texttt{Cover}(\ell_k^r(p), \bigcup_{\varphi_{ij}(a) \in \mathcal{E}_{k-1} : p \in \varepsilon_{ij}^+(a)} c_{k-1}(\varphi_{ij}(a)))$. The cost $c_k^r(p)$ in a set of states $\ell_k^r(p)$ for a proposition at level k is found by covering the states $\ell_k^r(p)$ with the union of all state-cost pairs of effects that support the proposition.

$$\begin{aligned}
c_1(\texttt{at(alpha)}) &= \{(\{s_\alpha, s_\beta, s_\gamma\}, 0)\} \\
c_1(\texttt{at(beta)}) &= \{(\{s_\alpha, s_\beta, s_\gamma\}, 10)\} \\
c_1(\texttt{at(gamma)}) &= \{(\{s_\alpha, s_\beta, s_\gamma\}, 30)\} \\
c_1(\texttt{have(soil)}) &= \{(\{s_\alpha\}, 20)\} \\
c_1(\texttt{avail(soil, alpha)}) &= \{(\{s_\alpha\}, 0)\} \\
c_1(\texttt{avail(soil, beta)}) &= \{(\{s_\beta\}, 0)\} \\
c_1(\texttt{avail(soil, gamma)}) &= \{(\{s_\gamma\}, 0)\}
\end{aligned}$$

Again, the cost vector for each proposition contains a single element. The propositions from level zero have no new supporters, so their cost vectors contain no new elements. The new propositions in level one have a single cost vector element because all worlds that label them are new. Each element of each cost vector is covered by a single element from their single supporter.

$$\begin{aligned}
c_1(\texttt{sample(soil, alpha)}) &= c_1(\texttt{drive(alpha, beta)}) &&= \\
c_1(\texttt{drive(alpha, gamma)}) &= c_1(\texttt{at(alpha)}_p) &&= \\
c_1(\texttt{commun(soil)}) &= c_1(\texttt{at(alpha)}_p) &&= (\{s_\alpha, s_\beta, s_\gamma\}, 0)
\end{aligned}$$

$$c_1(\texttt{drive(beta, gamma)}) = c_1(\texttt{sample(soil, beta)}) =$$
$$c_1(\texttt{at(beta)}_p) = c_1(\texttt{drive(beta, alpha)}) = (\{s_\alpha, s_\beta, s_\gamma\}, 10)$$
$$c_1(\texttt{at(gamma)}_p) = c_1(\texttt{drive(gamma, alpha)}) =$$
$$c_1(\texttt{drive(gamma, beta)}) = c_1(\texttt{sample(soil, gamma)}) = (\{s_\alpha, s_\beta, s_\gamma\}, 30)$$
$$c_1(\texttt{have(soil)}_p) = \{(\{s_\alpha\}, 20)\}$$
$$c_1(\texttt{avail(soil, alpha)}_p) = \{(\{s_\alpha\}, 0)\}$$
$$c_1(\texttt{avail(soil, beta)}_p) = \{(\{s_\beta\}, 0)\}$$
$$c_1(\texttt{avail(soil, gamma)}_p) = \{(\{s_\gamma\}, 0)\}$$

Each action in $\mathcal{A}_1$ has a single element in its cost vector because the action is new, or is supported in no new worlds. The new actions at this level have a non zero cost for the element in their cost vector because their preconditions have non zero cost. For example, `sample(soil, gamma)` has cost 30 for its single cost vector element because its precondition `at(gamma)` has cost 30 for its single cost vector element.

Continuing to $\mathcal{E}_1$, the cost vectors are:

$$c_1(\varphi_{00}(\texttt{drive(alpha, beta)})) = c_1(\varphi_{00}(\texttt{at(beta)}_p)) = \{(\{s_\alpha, s_\beta, s_\gamma\}, 10)\}$$
$$c_1(\varphi_{00}(\texttt{drive(alpha, gamma)})) = c_1(\varphi_{00}(\texttt{at(gamma)}_p)) = \{(\{s_\alpha, s_\beta, s_\gamma\}, 30)\}$$
$$c_1(\varphi_{00}(\texttt{drive(beta, alpha)})) = \{(\{s_\alpha, s_\beta, s_\gamma\}, 15)\}$$
$$c_1(\varphi_{00}(\texttt{drive(beta, gamma)})) = \{(\{s_\alpha, s_\beta, s_\gamma\}, 25)\}$$
$$c_1(\varphi_{00}(\texttt{drive(gamma, alpha)})) = \{(\{s_\alpha, s_\beta, s_\gamma\}, 35)\}$$
$$c_1(\varphi_{00}(\texttt{drive(gamma, beta)})) = \{(\{s_\alpha, s_\beta, s_\gamma\}, 45)\}$$
$$c_1(\varphi_{00}(\texttt{at(alpha)}_p)) = \{(\{s_\alpha, s_\beta, s_\gamma\}, 0)\}$$

$$
\begin{aligned}
&c_1(\varphi_{00}(\texttt{commun(soil)})) && &&= \{(\{s_\alpha\}, 25)\}\\
&c_1(\varphi_{00}(\texttt{sample(soil, alpha)})) &&= c_1(\varphi_{00}(\texttt{have(soil)}_p)) &&= \{(\{s_\alpha\}, 20)\}\\
&c_1(\varphi_{00}(\texttt{avail(soil, alpha)}_p)) && &&= \{(\{s_\alpha\}, 0)\}\\
&c_1(\varphi_{00}(\texttt{sample(soil, beta)})) && &&= \{(\{s_\beta\}, 30)\}\\
&c_1(\varphi_{00}(\texttt{avail(soil, beta)}_p)) && &&= \{(\{s_\beta\}, 0)\}\\
&c_1(\varphi_{00}(\texttt{sample(soil, gamma)})) && &&= \{(\{s_\gamma\}, 50)\}\\
&c_1(\varphi_{00}(\texttt{avail(soil, gamma)}_p)) && &&= \{(\{s_\gamma\}, 0)\}
\end{aligned}
$$

Each effect in level one has a single element in its cost vector because each is new or is labeled by no new worlds. The costs of each element reflects the cost of executing the associated action and the cost of supporting the action for all worlds represented by the element. For example, $\varphi_{00}(\texttt{sample(soil, beta)})$ has cost 30 for the element $\{s_\beta\}$ because executing `sample(soil, beta)` costs 20, supporting the action costs 10, and supporting the antecedent of the effect costs 0.

Continuing to $\mathcal{P}_2$, the cost vectors are:

$$
\begin{aligned}
c_2(\texttt{at(alpha)}) &= \{(\{s_\alpha, s_\beta, s_\gamma\}, 0)\}\\
c_2(\texttt{at(beta)}) &= \{(\{s_\alpha, s_\beta, s_\gamma\}, 10)\}\\
c_2(\texttt{at(gamma)}) &= \{(\{s_\alpha, s_\beta, s_\gamma\}, 25)\}\\
c_2(\texttt{have(soil)}) &= \{(\{s_\alpha\}, 20), (\{s_\beta, s_\gamma\}, 80)\}\\
c_2(\texttt{avail(soil, alpha)}) &= \{(\{s_\alpha\}, 0)\}\\
c_2(\texttt{avail(soil, beta)}) &= \{(\{s_\beta\}, 0)\}\\
c_2(\texttt{avail(soil, gamma)}) &= \{(\{s_\gamma\}, 0)\}\\
c_2(\texttt{comm(soil)}) &= \{(\{s_\alpha\}, 25)\}
\end{aligned}
$$

At level two, there is an example of a cost vector with more than one element. The `have(soil)` proposition was reachable by $\{s_\alpha\}$ at level one, and is newly reachable by $\{s_\beta, s_\gamma\}$ at this level. The first can be supported in the same fashion as in level one. The second element must be covered by supporting effect cost vectors. The element from $c_1(\varphi_{00}(\texttt{sample(soil, beta)}))$ has cost 30 and covers s_β and the cost vector of $\varphi_{00}(\texttt{sample(soil, gamma)})$ at level one has cost 50 and covers s_γ, making the cost of covering $\{s_\beta, s_\gamma\}$ a total of 80.

Notice also that the cost of `at(gamma)` decreases because its minimal cost supporter is now the effect of `drive(beta, gamma)`, instead of `drive(alpha, gamma)`. By propagating costs I can identify if longer action sequences reduce the cost of supporting propositions.

Continuing to $\mathcal{A}_2$, the cost vectors are:

$$\begin{aligned}
&c_2(\texttt{sample(soil, alpha)}) = c_2(\texttt{drive(alpha, beta)}) &&= \\
&c_2(\texttt{drive(alpha, gamma)}) = c_2(\texttt{at(alpha)}_p) &&= \\
&c_2(\texttt{commun(soil)}) = c_2(\texttt{at(alpha)}_p) &&= (\{s_\alpha, s_\beta, s_\gamma\}, 0) \\
&c_2(\texttt{drive(beta, gamma)}) = c_2(\texttt{sample(soil, beta)}) &&= \\
&c_2(\texttt{at(beta)}_p) = c_2(\texttt{drive(beta, alpha)}) &&= (\{s_\alpha, s_\beta, s_\gamma\}, 10) \\
&c_2(\texttt{at(gamma)}_p) = c_2(\texttt{drive(gamma, alpha)}) &&= \\
&c_2(\texttt{drive(gamma, beta)}) = c_2(\texttt{sample(soil, gamma)}) &&= (\{s_\alpha, s_\beta, s_\gamma\}, 25) \\
&c_2(\texttt{have(soil)}_p) &&= \{(\{s_\alpha\}, 20), \\
& && \quad (\{s_\beta, s_\gamma\}, 80)\} \\
&c_2(\texttt{comm(soil)}_p) &&= \{(\{s_\alpha\}, 25)\} \\
&c_2(\texttt{avail(soil, alpha)}_p) &&= \{(\{s_\alpha\}, 0)\} \\
&c_2(\texttt{avail(soil, beta)}_p) &&= \{(\{s_\beta\}, 0)\} \\
&c_2(\texttt{avail(soil, gamma)}_p) &&= \{(\{s_\gamma\}, 0)\}
\end{aligned}$$

The actions in level two have their cost vectors computed as before. Because the cost of at(gamma) decreased, the cost of sample(soil, gamma) also decreases to 25.

Continuing to $\mathcal{E}_2$, the cost vectors are:

$$\begin{aligned}
&c_2(\varphi_{00}(\texttt{drive(alpha, beta)})) = c_2(\varphi_{00}(\texttt{at(beta)}_p)) &&= \{(\{s_\alpha, s_\beta, s_\gamma\}, 10)\}\\
&c_2(\varphi_{00}(\texttt{drive(alpha, gamma)})) = &&= \{(\{s_\alpha, s_\beta, s_\gamma\}, 30)\}\\
&c_2(\varphi_{00}(\texttt{at(gamma)}_p)) &&= \{(\{s_\alpha, s_\beta, s_\gamma\}, 25)\}\\
&c_2(\varphi_{00}(\texttt{drive(beta, alpha)})) &&= \{(\{s_\alpha, s_\beta, s_\gamma\}, 15)\}\\
&c_2(\varphi_{00}(\texttt{drive(beta, gamma)})) &&= \{(\{s_\alpha, s_\beta, s_\gamma\}, 25)\}\\
&c_2(\varphi_{00}(\texttt{drive(gamma, alpha)})) &&= \{(\{s_\alpha, s_\beta, s_\gamma\}, 30)\}\\
&c_2(\varphi_{00}(\texttt{drive(gamma, beta)})) &&= \{(\{s_\alpha, s_\beta, s_\gamma\}, 35)\}\\
&c_2(\varphi_{00}(\texttt{at(alpha)}_p)) &&= \{(\{s_\alpha, s_\beta, s_\gamma\}, 0)\}\\
&c_2(\varphi_{00}(\texttt{commun(soil)})) &&= \{(\{s_\alpha\}, 25),\\
& && \quad (\{s_\beta, s_\gamma\}, 85)\}\\
&c_2(\varphi_{00}(\texttt{sample(soil, alpha)})) = c_2(\varphi_{00}(\texttt{have(soil)}_p)) &&= \{(\{s_\alpha\}, 20)\}\\
&c_2(\varphi_{00}(\texttt{avail(soil, alpha)}_p)) &&= \{(\{s_\alpha\}, 0)\}\\
&c_2(\varphi_{00}(\texttt{sample(soil, beta)})) &&= \{(\{s_\beta\}, 30)\}\\
&c_2(\varphi_{00}(\texttt{avail(soil, beta)}_p)) &&= \{(\{s_\beta\}, 0)\}\\
&c_2(\varphi_{00}(\texttt{sample(soil, gamma)})) &&= \{(\{s_\gamma\}, 45)\}\\
&c_2(\varphi_{00}(\texttt{avail(soil, gamma)}_p)) &&= \{(\{s_\gamma\}, 0)\}\\
&c_2(\varphi_{00}(\texttt{comm(soil)}_p)) &&= \{(\{s_\alpha\}, 25)\}
\end{aligned}$$

The effects at level two have their cost vectors computed as before. Notice that the cost of φ_{00}(sample(soil, gamma)) decreases to 45 from 50 in the last level. The cost decreases because the associated action had its cost decrease. Also, the effect φ_{00}(commun(soil, gamma)) has an additional element in its cost vector at this level because its condition `have(soil)` acquired a new cost vector element.

Continuing to $\mathcal{P}_3$, the cost vectors are:

$$\begin{array}{lcl}
c_3(\texttt{at(alpha)}) & = & \{(\{s_\alpha, s_\beta, s_\gamma\}, 0)\} \\
c_3(\texttt{at(beta)}) & = & \{(\{s_\alpha, s_\beta, s_\gamma\}, 10)\} \\
c_3(\texttt{at(gamma)}) & = & \{(\{s_\alpha, s_\beta, s_\gamma\}, 25)\} \\
c_3(\texttt{have(soil)}) & = & \{(\{s_\alpha\}, 20), (\{s_\beta, s_\gamma\}, 75)\} \\
c_3(\texttt{avail(soil, alpha)}) & = & \{(\{s_\alpha\}, 0)\} \\
c_3(\texttt{avail(soil, beta)}) & = & \{(\{s_\beta\}, 0)\} \\
c_3(\texttt{avail(soil, gamma)}) & = & \{(\{s_\gamma\}, 0)\} \\
c_3(\texttt{comm(soil)}) & = & \{(\{s_\alpha\}, 25), (\{s_\beta, s_\gamma\}, 85)\}
\end{array}$$

At level three, the goal `comm(soil)` is reachable from each possible world, and adding the cost vector elements indicates that the cost is 110. At this point, it is possible to find a relaxed plan to support the goal proposition by using the propagated cost information. It is also possible to extend the $CLUG$ additional levels to see if the cost of the goal propositions decreases. With costs, level-off does not occur until proposition costs do not change between levels, instead of when the proposition layers do not change (as before).

2.1. Relaxed Plans. The challenge to deal with cost-sensitive domains is in extracting cost-sensitive relaxed plans, for which the propagated cost vectors help. The relaxed plan heuristic extracted from the $CLUG$ is similar to the LUG relaxed plan heuristic, from the previous chapter. The only aspect of relaxed plan extraction that changes is how proposition supporters are chosen. In the LUG relaxed plans, the only information available about supporters is the number of worlds in which they support the proposition and whether or not they are persistence. Thus, without cost information, supporters that are persistence are preferred, as are supporters that cover more possible worlds. With cost information, it is possible to assess the cost with which supporters cover certain possible worlds and prefer the least costly.

In the LUG, I use an un-weighted set cover to cover the worlds in which a proposition should be supported. In the $CLUG$, it becomes a weighted set cover. As described in the `Cover` operation it is possible to greedily select supporters that cover new possible worlds based on the cost with which they cover new worlds. Consider the relaxed plan in Figure 27 depicted in bold. This relaxed plan is extracted using cost vectors for guidance, making it different from the relaxed plan extracted from the LUG in Figure 51. The primary difference is that the goal proposition `comm(soil)` is supported differently when cost information is present. The `comm(soil)` proposition can covered by three supporters φ_{00}(`commun(soil)`) in worlds $\{s_\alpha\}$ at cost 25, φ_{00}(`commun(soil)`) in worlds $\{s_\beta, s_\gamma\}$ at cost 75, and φ_{00}(`comm(soil)`$_p$) in worlds $\{s_\alpha\}$ at cost 25. Notice that φ_{00}(`commun(soil)`) is only one effect but is part of two supporters; each entry in the cost vector of an effect corresponds to a different way to support a proposition (because of different worlds and costs). In the LUG, the relaxed plan chose the latter two supporters because it chose the persistence first and needed the action effect to cover the other two worlds. In the $CLUG$, persistence is not arbitrarily chosen first. The relaxed plan extraction chooses the first supporter first because it has lowest cost (breaking the tie with persistence) and covers new worlds. Then, the second supporter is chosen because it is the only option for covering new worlds. The resulting relaxed plan has cost 105, where the relaxed plan extracted from the LUG (without using cost information) has cost 110 because it uses the `commun(soil)` action twice.

3. Empirical Evaluation

The main intent of this chapter is to evaluate the effectiveness of the $CLUG$ in improving the quality of plans generated by $POND$. Additionally, I also compare with two state of the art planners, GPT (Bonet and Geffner, 2000), and MBP (Bertoli *et al.*, 2001).[1] Even though MBP does not plan with costs, I show the cost of MBP's plans for each problem's cost model. GPT uses heuristics based on relaxing the problem to full-observability (whereas our relaxation is to no observability while ignoring negative interactions), and MBP uses a belief state's size as its heuristic merit.

The test set up involves two domains: Medical-Specialist and Rovers. Each problem had a time out of 20 minutes and a memory limit of 1GB on a 2.6GHz P4 Linux machine.

Medical-Specialist: I 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. I assigned costs as follows: c(stain) = 5, c(count_white_cells) = 10, c(inspect_stain) = X, c(analyze_white_cell_count) = X, c(medicate) = 5, and c(specialist_medicate) = 10. I generated ten problems, each with the respective number of diseases (1-10), in two sets where X = $\{15, 25\}$. Plans in this domain must treat a patient by either performing some combination of staining, counting white cells, and sensing actions to diagnose the exact disease and apply the proper medicate action, or using the specialist medicate action without knowing the exact disease. Plans can use hybrid strategies, using the specialist medicate for some diseases and the diagnosis and medicate for others. The strategy depends on cost and the number of diseases.

The results in Figures 52, 53, and 54 show the average plan path cost, number of plan nodes

[1]While there are other conditional planners, such as YAK (Rintanen, 2002) and SGP (Weld *et al.*, 1998), I chose MBP to represent those that are not cost sensitive and GPT for its ability to handle cost. In our other experiments, not shown, YAK and SGP are inferior to MBP anyway.

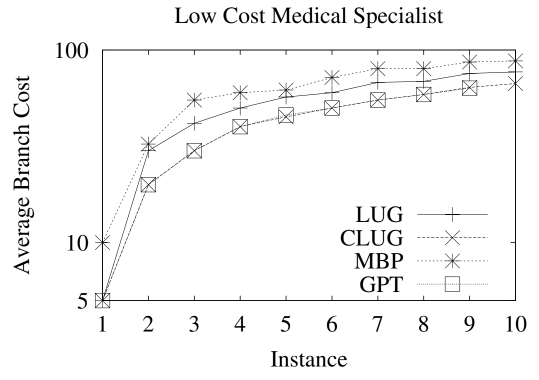


Figure 52. Quality (average plan path cost) for $POND$ (LUG and $CLUG$), MBP, and GPT for Medical-Specialist.

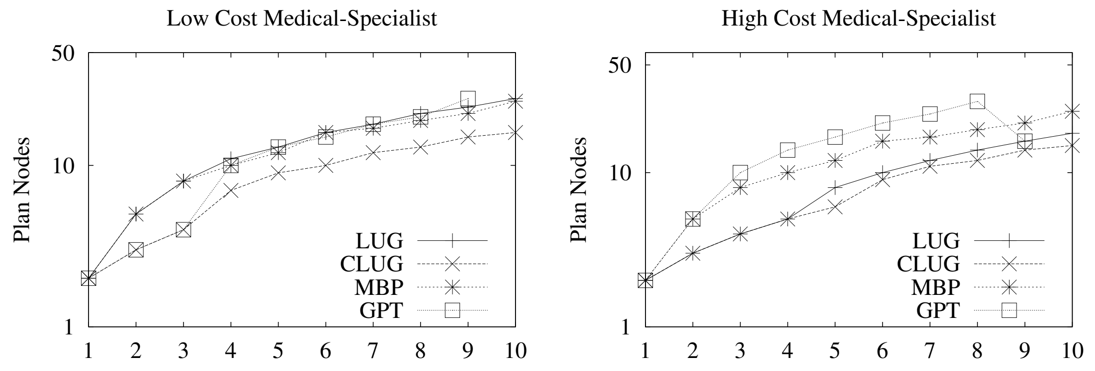


Figure 53. Number of plan nodes for $POND$ (LUG and $CLUG$), MBP, and GPT for Medical-Specialist.

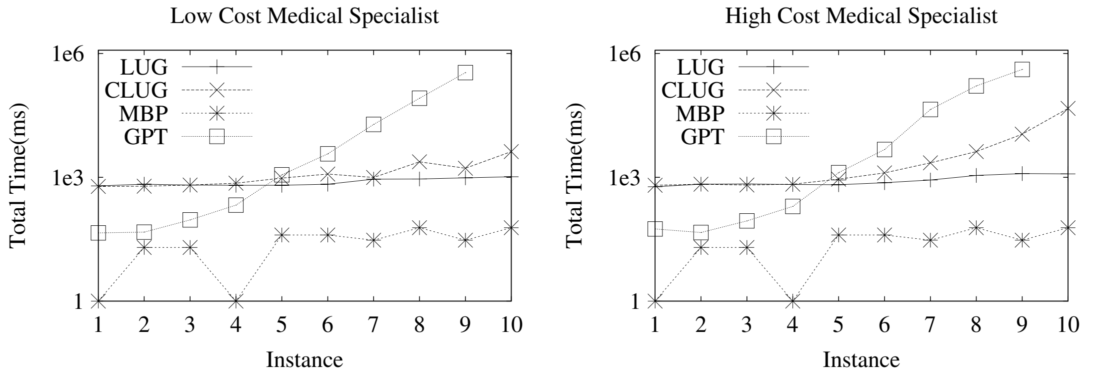


Figure 54. Total Time(ms) for $POND$ (LUG and $CLUG$), MBP, and GPT for Medical-Specialist.

(belief states) in the solution, and total time for two cost models; the horizontal axis reflects different problem instances. Extracting relaxed plans from the $CLUG$ instead of the LUG enables $POND$ to be more cost-sensitive. The plans returned by the $CLUG$ method tend to have less nodes and a lower average path cost than the LUG. The LUG heuristic does not measure sensing cost, but as sensing cost changes, the search is able to locally gauge the cost of sensing and adapt. Since MBP is insensitive to cost, 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 may be attributed to how the heuristic is computed by relaxing the problem to full-observability. The LUG and $CLUG$ heuristics measure the cost of co-achieving the goal from a set of states, whereas GPT takes the average cost for reaching the goal from the states.

Rovers: I use an adaptation of the Rovers domain from the previous chapter. The action cost model is: c(sense_visibility) = X, c(sense_rock) = Y, c(sense_soil) = Z, c(navigate) = 50, c(calibrate) = 10, c(take_image) = 20, c(communicate_data) = 40, c(sample_soil) = 30, c(sample_rock) = 60, and c(drop) = 5. The two versions have costs: (X,Y,Z) = $\{$(35, 55, 45), (100, 120, 110)$\}$. Plans in the rovers domain can involve sensing at locations to identify if data can be collected or simply going to every possible location and trying to collect data. The number of locations varies between four and eight, and the number of possible locations to collect up to three types of data can be between one and four.

The results in Figures 55, 56, and 57 show the average plan path cost, number of plan nodes (belief states) in the solution, and total time for two cost models; the horizontal axis reflects different problem instances. The LUG and $CLUG$ relaxed plan extraction guide $POND$ toward similar plans, in terms of cost and number of nodes. The lack of difference between the heuristics may be attributed to the domain structure – good solutions have a lot of positive interaction (i.e. the heuristics extract similar relaxed plans because low cost actions also support subgoals in many pos-

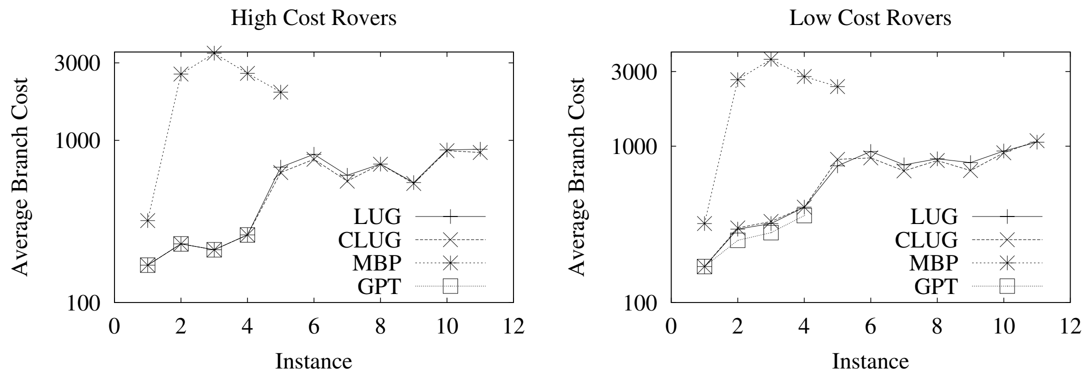


Figure 55. Quality (average plan path cost) for $POND$ (LUG and $CLUG$), MBP, and GPT for Rovers.

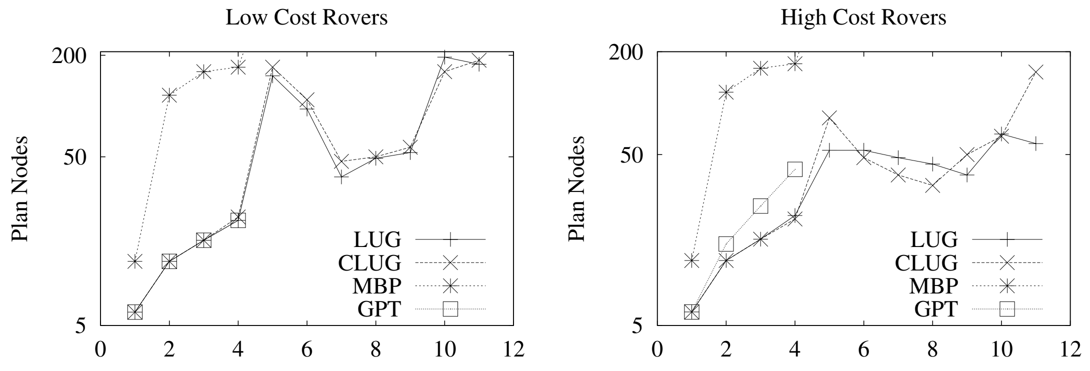


Figure 56. Number of plan nodes for $POND$ (LUG and $CLUG$), MBP, and GPT for Rovers.

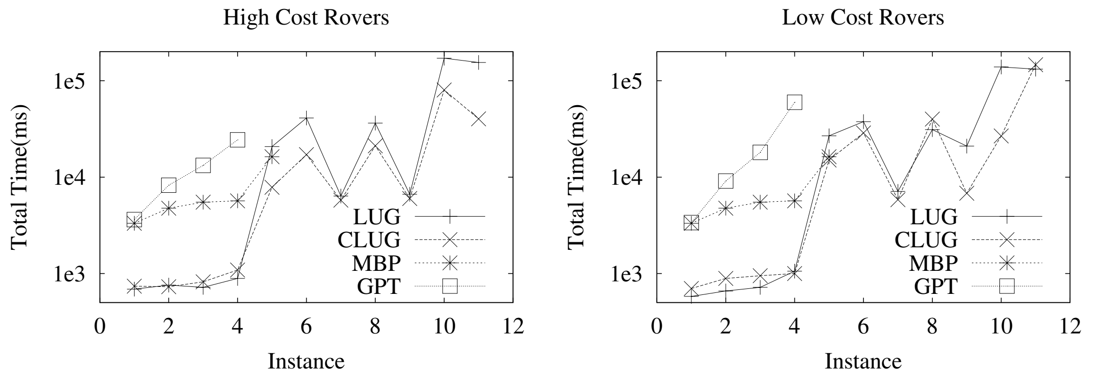


Figure 57. Total Time(ms) for $POND$ (LUG and $CLUG$), MBP, and GPT for Rovers.

sible worlds), as opposed to Medical-Specialist where solutions are fairly independent for different possible worlds. MBP, making no use of action costs, returns plans with considerably (a order of magnitude) higher average path costs and number of solution nodes. GPT fares better than MBP in terms of plan cost, but both are limited in scalability due to weaker heuristics.

In summary, the experiments show that using the $CLUG$ to extract relaxed plans can help find better solutions. I also found that planners not reasoning about action cost can return arbitrarily poor solutions, and planners whose heuristic relaxes uncertainty do not scale as well.

4. Related Work

The idea of cost propagation on planning graphs was first presented by Do and Kambhampati (2003) to cope with metric-temporal planning. The first work on using planning graphs in conditional planning was in the CGP (Smith and Weld, 1998) and SGP (Weld *et al.*, 1998) planners. Recently, planning graph heuristics have proven useful in conformant planning (Bryce and Kambhampati, 2004; Brafman and Hoffmann, 2004) and conditional planning (Cushing and Bryce, 2005; Hoffmann and Brafman, 2005), as seen in the previous chapters. They have also proven useful in reachability analysis for MDPs (Boutilier *et al.*, 1999); this work could be extended for POMDPs. Also related is the work on sensor planning, such as Koenig and Liu (1999). The authors investigate the frequency of sensing as the plan optimization criterion changes (from minimizing the worst case cost to the expected cost). I investigate the frequency of sensing while minimizing average plan cost under different cost models. The work on optimal limited contingency planning (Meuleau and Smith, 2003) stated that adjusting sensory action cost, as we have, is an alternative to their approach for reducing plan branches.

5. Conclusion

With motivation toward conditional planning approaches that can scale like classical planners, but still reason with quality metrics like POMDPs, I have presented a cost propagated version of the LUG called the $CLUG$. With the $CLUG$ it is possible extract cost-sensitive relaxed plans that are effective in guiding $POND$ toward high-quality conditional plans. I have shown with an empirical comparison that my approach improves the quality of conditional plans over conditional planners that do not account for cost information, and that it can out-scale approaches that consider cost information and uncertainty in a weaker fashion.

CHAPTER 6

UNCERTAIN ACTION EFFECTS IN LABELED PLANNING GRAPHS

This chapter describes how to extend the *LUG* to address probabilistic conformant planning problems with actions whose effects are uncertain. Previous chapters address non-deterministic planning problems whose action effects are deterministic. I simultaneously move to probabilistic problems and actions with uncertain effects (i.e., conformant probabilistic planning) because the techniques described here make use of both assumptions. Unlike the previous chapter, this chapter assumes actions have unit cost.

1. Introduction

Despite long standing interest (Kushmerick *et al.*, 1994; Majercik and Littman, 1998; Hyafil and Bacchus, 2003, 2004), probabilistic plan synthesis algorithms have a terrible track record in terms of scalability. The current best conformant probabilistic planners are only able to handle very small problems. In contrast, there has been steady progress in scaling deterministic and non-deterministic planning. Much of this progress has come from the use of sophisticated reachability heuristics (as described in earlier chapters). In this chapter, I show how to effectively use reachability heuristics to solve probabilistic conformant planning problems. I use work on planning graph heuristics for non-deterministic planning described in previous chapters as the starting point. The heuristics developed in this chapter significantly improve the scalability of probabilistic conformant planning.

I investigate an extension of the labeled uncertainty graph to handle actions with probabilistic effects. Recall that the *LUG* is used to symbolically represent a set of relaxed planning graphs (much like the multiple planning graphs used by Conformant GraphPlan (Smith and Weld, 1998)), where each is associated with a possible world. While the *LUG* previously only worked with state uncertainty (and deterministic actions), it is necessary in probabilistic conformant planning to handle action uncertainty. Extending the *LUG* to consider action uncertainty involves symbolically representing how at each level the explicit representation of multiple planning graphs splits each planning graph over all joint outcomes of uncertain actions. In such a case, each time step has a set of planning graph layers (each a possible world) defined by the cross product of an exponential set of joint action outcomes and an exponential number of possible worlds from the previous level.

With deterministic actions, the *LUG* worked well because while there were an exponential number of possible worlds at each time step, the number was held constant (i.e., all uncertainty is in the starting belief state). With uncertain actions, an explicit or symbolic representation of planning graphs for all possible worlds at each time step is *exactly* representing an exponentially increasing set. Since I am only interested in planning graphs to compute heuristics, it is both impractical and unnecessary to exactly represent all of the reachable possible worlds. I turn to approximate methods for representing the possible worlds. Since I am applying planning graphs in a probabilistic setting, I have the opportunity to use Monte Carlo techniques to sample the planning graphs I construct. Unlike chapter 3, where I described uniform sampling from non-deterministic belief states to construct the planning graph, Monte Carlo relies on having probability distributions in belief states and action outcomes.

There are a wealth of methods that fall under the name sequential Monte Carlo (SMC) (Arulampalam *et al.*, 2002; Doucet *et al.*, 2001), for reasoning about a random variable over time. The idea in SMC is to represent a probability distribution as a set of samples (particles), which are updated over sequential time steps by sampling the system's transition function (i.e., simulating each particle). In this setting, each particle is a planning graph that represents a simulation in the relaxed planning space. Instead of splitting over all joint outcomes of uncertain actions to represent the true distribution over possible worlds in the planning graph, I sample a joint outcome of the actions to simulate the planning graph. By using more particles, I capture more of the possible worlds, exploiting the natural affinity between SMC approximation and heuristic accuracy.

The SMC technique requires multiple planning graphs (each a particle), but their number is fixed. I could represent each planning graph explicitly (analogous to the multiple graph techniques described earlier), but they may have considerable redundant structure. Instead, I generalize the *LUG* to symbolically represent the set of planning graph particles in a planning graph I call the Monte Carlo *LUG* ($\mathcal{M}c$*LUG*). I show that by using the $\mathcal{M}c$*LUG* to extract a relaxed plan heuristic $POND$ is able to greatly out-scale the current best conformant probabilistic planner CPplan (Hyafil and Bacchus, 2004, 2003) in a number of domains, without giving up too much in terms of plan quality.

My presentation starts with a brief primer on SMC. I follow with a worked example of how to construct planning graphs that exactly compute the probability distribution over possible worlds versus using SMC, as well as how one would symbolically represent planning graph particles. After the intuitive example, I give the details of the $\mathcal{M}c$*LUG* and the associated relaxed plan heuristic. Finally, I present an empirical analysis of my technique compared to CPplan, a discussion of related work, and conclusions.

2. Sequential Monte Carlo

In many scientific disciplines it is necessary to track the distribution over values of a random variable X over time. This problem can be stated as a first-order stationary Markov process with an initial distribution $p(X_0)$ and transition equation $p(X_{k+1}|X_k)$. It is possible to compute the probability distribution over the values of X after k steps as:

$$p(X_{k+1}) = \prod_{i=0}^{k} p(X_{i+1}|X_i)p(X_0) = p(X_{k+1}|X_k)...p(X_2|X_1)p(X_1|X_0)p(X_0)$$

In general, $p(X_k)$ can be very difficult to compute exactly.

SMC techniques can approximate $p(X_k)$ as a set of N samples (particles) $\{x_k^n\}_{n=0}^{N-1}$, where the probability that X_k takes value x_k,

$$P(X_k = x_k) \approx \frac{|\{x_k^n | x_k^n = x_k\}|}{N}$$

is the proportion of particles taking on value x_k. At time $k = 0$, the set of samples is drawn from the initial distribution $p(X_0)$. At each time step $k > 0$, SMC simulates each particle from time k by sampling the transition equation $x_{k+1}^n \sim p(X_{k+1} | x_k^n)$. Applying SMC to planning graphs, samples represent possible worlds and the transition equation resembles the Conformant GraphPlan (Smith and Weld, 1998) construction semantics.

I would like to point out that the SMC technique is inspired by, but different from the standard particle filter. The difference is that the SMC is used for prediction and not on-line filtering. It is not used to filter observations to weight particles for re-sampling. Particles are assumed to be unit weight throughout simulation.

3. Uncertain Effects in Planning Graphs

This section discusses how to incorporate reasoning about uncertain effects within planning graphs by using an extension of the rover example, shown in Figure 58. Like the description in chapter 2, there are probabilistic effects of the `sample` and `commun` actions, but there are no observations. I will start by describing how a direct extension of multiple planning graphs and the *LUG* might capture all possible worlds, from both initial state and action uncertainty. In the next section I will discuss how SMC can be used to reduce the representational burden of reasoning about the immense number of possible worlds.

It is possible to extend multiple planning graphs and the *LUG* to handle probabilities by associating probabilities with each planning graph or label. However, handling uncertain actions,

```
(define (domain rovers_stochastic)
  (:requirements :strips :typing)
  (:types location data)
  (:predicates
           (at ?x - location)
           (avail ?d - data ?x - location)
           (comm ?d - data)
           (have ?d - data))
  (:action drive
   :parameters (?x ?y - location)
   :precondition (at ?x)
   :effect (and (at ?y) (not (at ?x))))
  (:action commun
   :parameters (?d - data)
   :precondition (and)
   :effect (when (have ?d)
                 (probabilistic 0.8 (comm ?d))))
  (:action sample
   :parameters (?d - data ?x - location)
   :precondition (at ?x)
   :effect (when (avail ?d ?x)
                 (probabilistic 0.9 (have ?d)))))
)
```

```
(define (problem rovers_stochastic1)
  (:domain rovers)
  (:objects
     soil image rock - data
     alpha beta gamma - location)
  (:init (at alpha)
        (probabilistic 0.4 (avail soil alpha)
                       0.5 (avail soil beta)
                       0.1 (avail soil gamma))
  (:goal (comm soil) 0.5)
)
```

Figure 58. PDDL description of stochastic planning formulation of rover problem.

whether non-deterministic or stochastic, is troublesome. With deterministic actions, each of the multiple planning graphs or labels captured uncertainty about the source belief state and the number of planning graphs and size of the labels is bounded (there is a finite number of states in the belief state). With uncertain actions, multiple planning graphs and labels must capture uncertainty about the belief state and *each uncertain action at each level of the planning graph* because every execution of an action may have a different result.

Multiple Planning Graphs: Consider Figure 59, where both multiple planning graphs and the *LUG* are used to exactly represent possible worlds for the rover example. On the left of the figure, there are three initial proposition layers, as we saw when describing multiple planning graphs. However, since the `sample(soil, alpha)` action now has two outcomes (one with a conditional effect, and the other with no effect), there are different proposition layers reachable depending on the

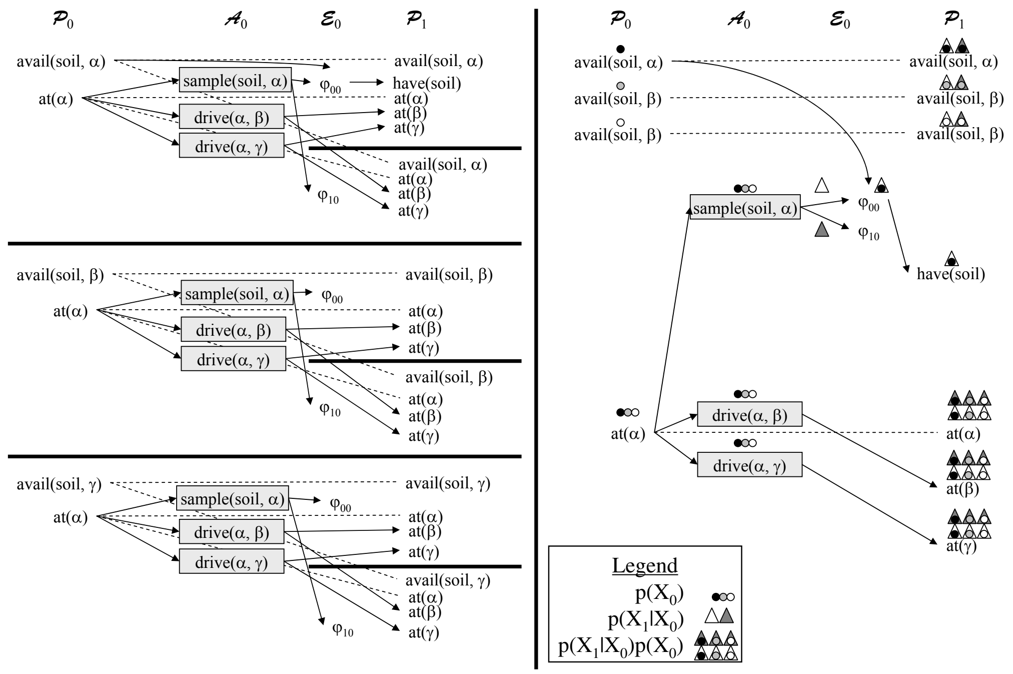


Figure 59. Multiple planning graphs (left) and Labeled uncertainty graph (right) with uncertain effects.

outcome. Each of the planning graphs "split" in the zeroth effect layer over all joint outcomes of uncertain actions. Since there is only one uncertain action (with two outcomes) in each level zero, each planning graph splits in two. If there were two uncertain actions (each with two outcomes), then each graph would split into four (one for each joint outcome of the actions). Since each of the initial proposition layers represents state uncertainty, each proposition layer is the value of a random variable:

$X_0 = \{\mathcal{P}_0(\{s_\alpha\}), \mathcal{P}_0(\{s_\beta\}), \mathcal{P}_0(\{s_\gamma\})\}.$

The next random variable:

$$\begin{aligned}X_1 = \{&\mathcal{P}_1(\{s_\alpha, \Phi_0(\texttt{sample(soil, alpha)})\}),\\ &\mathcal{P}_1(\{s_\alpha, \Phi_1(\texttt{sample(soil, alpha)})\}),\\ &\mathcal{P}_1(\{s_\beta, \Phi_0(\texttt{sample(soil, alpha)})\}),\\ &\mathcal{P}_1(\{s_\beta, \Phi_1(\texttt{sample(soil, alpha)})\}),\\ &\mathcal{P}_1(\{s_\gamma, \Phi_0(\texttt{sample(soil, alpha)})\}),\\ &\mathcal{P}_1(\{s_\gamma, \Phi_1(\texttt{sample(soil, alpha)})\})\}\end{aligned}$$

has a value for each proposition layer reachable from the joint outcome of all actions in action layer zero. Thus, after level one, the planning graphs capture the distribution $p(X_1|X_0)p(X_0)$.

Labeled Uncertainty Graph: As seen with multiple planning graphs with deterministic effects, there is considerable redundance among the multiple planning graphs. With uncertain effects, the redundancy increases as the planning graphs split. The ideas from the *LUG* can be extended to symbolically capture the possible worlds by using a new labeling scheme. In the problems considered in previous chapters, there is only a single random event X_0 captured by labels because the actions are deterministic. In the worst case, the random event X_0 captured by the labels has $2^{|P|}$ outcomes (i.e., all states are in the belief state), characterized by a logical formula over $\log_2(2^{|P|}) = |P|$ boolean variables.

When there are uncertain actions, the distribution $p(X_1|X_0)p(X_0)$ requires additional variables to represent $p(X_1|X_0)$. For example, if the action layer contains $|A|$ actions, each with m probabilistic outcomes, then we would require an additional $\log_2(m^{|A|}) = |A|\log_2(m)$ boolean variables (for a total of $|P| + |A|\log_2(m)$ boolean variables to exactly represent the distribution $p(X_1|X_0)p(X_0)$). For the distribution after k steps, it would need $|P| + k|A|\log_2(m)$ boolean variables. In a reasonable sized domain, where $|P| = 20$, $|A| = 30$, and $m = 2$, a LUG with $k = 3$ steps could require $20+(3)30\log_2(2) = 110$ boolean variables, and for $k = 5$ it needs 170. Currently, a label function with this many boolean variables is feasible to construct, but is too costly to use in

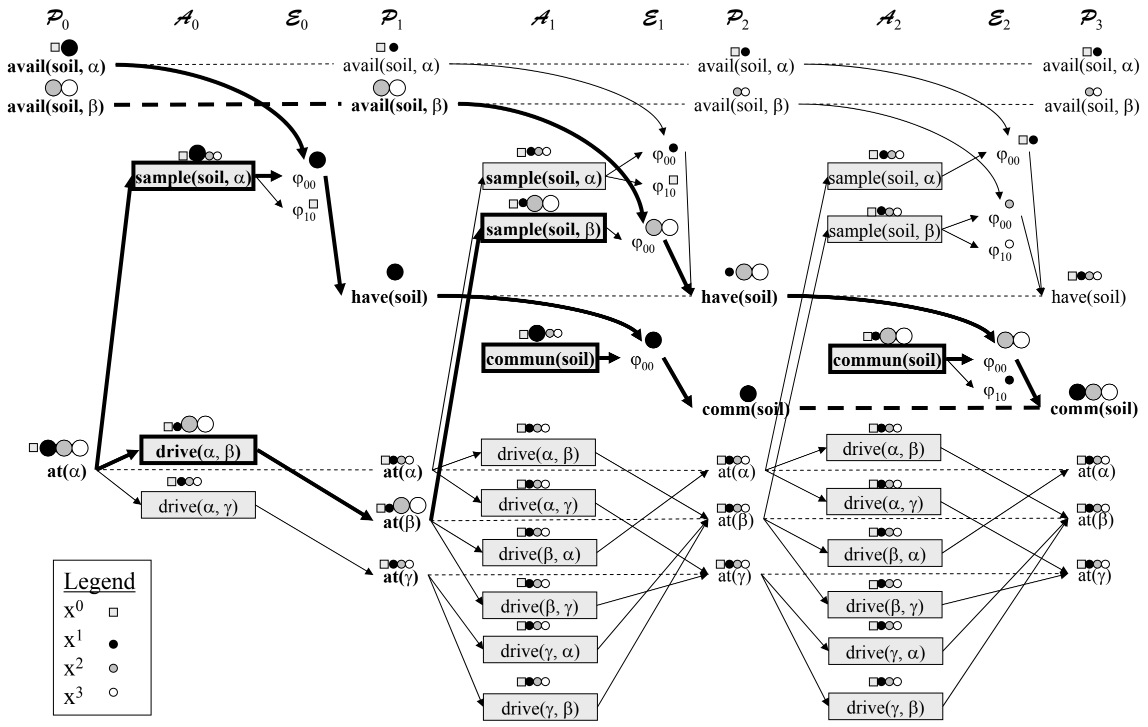


Figure 60. Monte carlo labeled uncertainty graph.

heuristic computation. I implemented this approach (representing labels as BDDs) and it performed very poorly; in particular it ran out of memory constructing the first planning graph for the p2-2-2 logistics problem, described in Section 5.

The planning graph on the right of Figure 59 shows a version of the *LUG* where the effect layer introduces label extensions to capture uncertain action outcomes. Where the zeroth proposition layer had three possibilities captured by each label, the level one proposition layer has six possibilities. Since each of the three states could have either outcome of `sample(soil, alpha)` occur, there is a possible world for each. Unlike the case with deterministic actions, the label representation grows (exponentially) with each level that contains uncertain actions. This growth can become too burdensome to represent even with powerful symbolic techniques. However, it is possible to sample the possible worlds to represent.

4. Monte Carlo in Planning Graphs

Recall from the previous chapters how I described sampling some of the states in a belief state to use for computing a heuristic. It is also possible to sample from the multiple planning graphs and *LUG* that incorporate uncertain action effects. Viewing the multiple planning graphs as a tree-like structure that branches over uncertainty, sampling will identify a subset of the paths to use for heuristic computation. The same idea can be extend within the *LUG*. In a probabilistic setting, it is possible to sample from the probability distribution over the multiple planning graphs to identify the most likely paths in the multiple planning graph tree. Using SMC, many of the paths may be sampled by the same particles, meaning they are more likely.

The Monte Carlo *LUG* ($\mathcal{M}c$*LUG*) represents every planning graph sample simultaneously using the labeling technique developed in the *LUG*. Figure 60 depicts a $\mathcal{M}c$*LUG* for the initial belief state of the example. There are four samples (particles), denoted by the circles and square above each action and proposition. Each effect is labeled by particles (sometimes with less particles than the associated action). The first two particles sample the first state s_α in the belief state, and the latter two sample the second state s_β. The third state s_γ, whose probability is 0.1, is not sampled, thus `avail(soil, gamma)` does not appear in $\mathcal{P}_0$. Every action that is supported by $\mathcal{P}_0$ is added to $\mathcal{A}_0$. In $\mathcal{A}_0$, `sample(soil, alpha)` is labeled by two particles, but each outcome is labeled by only one. This is because each particle labeling an action samples an outcome of the action. It happens that only one of the particles labeling `sample(soil, alpha)` samples the outcome with an effect; the other particle samples the outcome with no effect. In each action level, the $\mathcal{M}c$*LUG* must re-sample which particles label each outcome because each execution of an action can have a different outcome. Propagation continues until the proportion of particles labeling the goal is no less than the goal probability threshold. In $\mathcal{P}_2$, `comm(soil)` is labeled with one particle, indicating its probability is approximately 1/4, which is less than the threshold 0.5. In $\mathcal{P}_3$, `comm(soil)` is labeled with three particles, indicating its probability is approximately 3/4, which is greater than 0.5. It is possible to extract a labeled relaxed plan (described in the previous section) to support the goal for the three particles that label it.

4.1. Symbolic Particle Representation ($\mathcal{M}cLUG$). In the following, I formally describe how to construct the $\mathcal{M}cLUG$, an extension of the LUG to handle actions with uncertain outcomes, which is used to extract relaxed plan heuristics. The $\mathcal{M}cLUG$ is a set of layers and a label function $\mathcal{M}cLUG(b) = \langle(\mathcal{P}_0, \mathcal{A}_0, \mathcal{E}_0, ..., \mathcal{A}_{k-1}, \mathcal{E}_{k-1}, \mathcal{P}_k), \ell\rangle$. There are noticeable similarities to the *LUG*, but by using a fixed number of particles the $\mathcal{M}cLUG$ avoids adding boolean variables to the label function at each level of the graph. I implement labels as boolean formulas, but find it convenient in this context to describe them as sets of particles (where each particle is in reality a model of a boolean formula). The $\mathcal{M}cLUG$ is constructed with respect to a belief state encountered in search, which I call the source belief state. The algorithm to construct the $\mathcal{M}cLUG$ starts by forming an initial proposition layer $\mathcal{P}_0$ and an inductive step to generate a graph level $\{\mathcal{A}_k, \mathcal{E}_k, \mathcal{P}_k\}$ consisting of an action, effect, and proposition layer. I describe each part of this procedure in detail and follow with a description of relaxed plan extraction.

Initial Proposition Layer: The initial proposition layer is constructed with a set of N particles $\{x_0^n\}_{n=0}^{N-1}$ drawn from the source belief state. Each particle x_0^n corresponds to a state $s \in b$ in the source belief state. (The super-script of a particle denotes its identity, and the sub-script denotes its time index.)

In the example (assuming N=4), the samples map to the states:

$$x_0^0 = x_0^1 = \{s_\alpha\},$$

$$x_0^2 = x_0^3 = \{s_\beta\}$$

The initial proposition layer $\mathcal{P}_0$ is a set of labeled propositions $\mathcal{P}_0 = \{p|\ell_0(p) \neq \emptyset\}$, where each proposition must be labeled with at least one particle. A proposition is labeled $\ell_0(p) = \{x_0^n|p \in s, x_0^n = s\}$ to denote particles that correspond to states where the proposition holds.

In the example, the initial proposition layer is:

$$\mathcal{P}_0 = \{\texttt{at(alpha), avail(soil, alpha), avail(soil, beta)}\}$$

and the labels are:

$$\begin{aligned}
\ell_0(\texttt{at(alpha)}) &= \{x_0^0, x_0^1, x_0^2, x_0^3\} \\
\ell_0(\texttt{avail(soil, alpha)}) &= \{x_0^0, x_0^1\} \\
\ell_0(\texttt{avail(soil, beta)}) &= \{x_0^2, x_0^3\}
\end{aligned}$$

Action Layer: The action layer at time k consists of all actions whose enabling precondition is enabled, meaning all of the enabling precondition propositions hold together in at least one particle. The action layer is defined as all enabled actions $\mathcal{A}_k = \{a|\ell_k(a) \neq \emptyset\}$, where the label of each action is the set of particles where it is enabled $\ell_k(a) = \bigcap_{p \in \rho^e(a)} \ell_k(p)$. When the enabling precondition is empty, the label contains all particles.

In the example, the zeroth action layer is:

$$\begin{aligned}
\mathcal{A}_0 = \ &\{\texttt{sample(soil, alpha), drive(alpha, beta),} \\
&\ \ \texttt{avail(soil, alpha)}_p\texttt{, avail(soil, beta)}_p\texttt{,} \\
&\ \ \texttt{at(alpha)}_p\}
\end{aligned}$$

$$\begin{aligned}
\ell_0(\texttt{sample(soil, alpha)}) = \ell_0(\texttt{drive(alpha, beta)}) &= \\
\ell_0(\texttt{drive(alpha, gamma)}) = \ell_0(\texttt{at(alpha)}_p) &= \{x_0^0, x_0^1, x_0^2, x_0^3\} \\
\ell_0(\texttt{avail(soil, alpha)}_p) &= \{x_0^0, x_0^1\} \\
\ell_0(\texttt{avail(soil, beta)}_p) &= \{x_0^2, x_0^3\}
\end{aligned}$$

Each of the actions are enabled in all particles because their enabling preconditions are $\{\}$, thus always enabled. The persistence actions for the `avail` proposition are labeled by the particles where they were sampled in the proposition layer.

Effect Layer: The effect layer contains all effects that are labeled with at least one particle, such that $\mathcal{E}_k = \{\varphi_{ij}(a) | \ell_k(\varphi_{ij}(a)) \neq \emptyset\}$. Determining which effects get labeled requires simulating the path of each particle. The path of a particle is simulated by sampling from the distribution over the joint outcomes of all enabled actions, $x^n_{k+1} \sim p(X_{k+1}|x^n_k)$. I sample by first identifying the actions $a^n_k = \{a | x^n_k \in \ell_k(a)\}$ that are applicable for a particle x^n_k. I sample from the distribution of outcomes of each applicable action $a \in a^n_k$ to identify a joint set of outcomes $\Phi^n_k = \{\Phi_i(a) | \Phi_i(a) \sim \Phi(a), a \in a^n_k\}$. The set of sampled outcomes identifies the path of x^n_k to x^n_{k+1}. I record the path by adding x^n_{k+1} to the labels $\ell_k(\varphi_{ij}(a))$ of applicable effects $\varphi_{ij}(a) \in \Phi_i(a)$ of sampled outcomes $\Phi_i(a) \in \Phi^n_k$. Note that even though an outcome is sampled for a particle, some of its effects may not be applicable because their antecedents are not supported by the particle (i.e., $x^n_k \notin \bigcap_{p \in \rho_{ij}(a)} \ell_k(p)$).

In the example, I first simulate x^0_0 by sampling the outcomes of all actions applicable in x^0_0, which is every action. Because the `drive` actions are deterministic, the same outcome will always be sampled, so I will ignore these actions in this description. Suppose outcome 1 for `sample(soil, alpha)` is sampled, meaning it is labeled with x^0_1. Particle x^1_0 happens to sample outcome 0, and it labels the effects of outcome 0. Particles x^2_0 and x^3_0 both sample outcome 0, but do not label the effect in outcome 0 because the effect does not have its antecedent enabled in x^2_0 and x^3_0.

Thus, the effect layer is:

$$\begin{aligned}\mathcal{E}_0 = \ &\{\varphi_{00}(\texttt{sample(soil, alpha)}), \varphi_{10}(\texttt{sample(soil, alpha)}),\\ &\varphi_{00}(\texttt{drive(alpha, beta)}), \varphi_{00}(\texttt{drive(alpha, gamma)}),\\ &\varphi_{00}(\texttt{avail(soil, alpha)}_p), \varphi_{00}(\texttt{avail(soil, beta)}_p),\\ &\varphi_{00}(\texttt{at(alpha)}_p)\}\end{aligned}$$

$$\begin{aligned}&\ell_0(\varphi_{00}(\texttt{drive(alpha, beta)})) = \ell_0(\varphi_{00}(\texttt{drive(alpha, gamma)}))=\\ &\ell_0(\varphi_{00}(\texttt{at(alpha)}_p)) &&=\{x_1^0, x_1^1, x_1^2, x_1^3\}\\ &\ell_0(\varphi_{00}(\texttt{sample(soil, alpha)})) &&=\{x_1^1\}\\ &\ell_0(\varphi_{10}(\texttt{sample(soil, alpha)})) &&=\{x_1^0\}\\ &\ell_0(\varphi_{00}(\texttt{avail(soil, alpha)}_p)) &&=\{x_1^0, x_1^1\}\\ &\ell_0(\varphi_{00}(\texttt{avail(soil, beta)}_p)) &&=\{x_1^2, x_1^3\}\end{aligned}$$

Proposition Layer: Proposition layer $\mathcal{P}_k$ contains all propositions that are given by an effect in $\mathcal{E}_{k-1}$. Each proposition is labeled by the particles of every effect that gives it support. The proposition layer is defined as $\mathcal{P}_k = \{p|\ell_k(p) \neq \emptyset\}$, where the label of a proposition is

$$\ell_k(p) = \bigcup_{\varphi_{ij}(a)\in\mathcal{E}_{k-1}:p\in\varepsilon_{ij}^+(a)} \ell_{k-1}(\varphi_{ij}(a))$$

In the example, the level one proposition layer is

$$\mathcal{P}_1 = \mathcal{P}_0 \cup \{\texttt{at(beta), at(gamma), have(soil)}\}$$

$$\begin{aligned}&\ell_0(\texttt{at(alpha)}) &&= \ \ell_0(\texttt{at(beta)}) \ = \\ &\ell_0(\texttt{at(gamma)}) &&= \ \{x_1^0, x_1^1, x_1^2, x_1^3\}\\ &\ell_0(\texttt{have(soil)}) &&= \ \{x_1^1\}\\ &\ell_0(\texttt{avail(soil, alpha)}) &&= \ \{x_1^0, x_1^1\}\\ &\ell_0(\texttt{avail(soil, beta)}) &&= \ \{x_1^2, x_1^3\}\end{aligned}$$

The propositions from the previous proposition layer $\mathcal{P}_0$ persist through noop actions, allowing them to be labeled as in the previous level – in addition to particles from any new supporters. The `have(soil)` proposition is supported by one effect, and one particle defines the label. The `drive` actions are support `at(beta)` and `at(gamma)` in all particles.

Continuing to the level one action layer:

$$\begin{aligned}\mathcal{A}_1 = \mathcal{A}_0 \cup\ \{&\texttt{sample(soil, beta)},\\ &\texttt{commun(soil), drive(beta, alpha)},\\ &\texttt{drive(beta, gamma), drive(gamma, alpha)},\\ &\texttt{drive(gamma, beta),have(soil)}_p\}\end{aligned}$$

$$\begin{array}{lcll}
\ell_1(\texttt{sample(soil, alpha)}) & = & \ell_1(\texttt{drive(alpha, beta)}) & = \\
\ell_1(\texttt{drive(alpha, gamma)}) & = & \ell_1(\texttt{at(alpha)}_p) & = \\
\ell_1(\texttt{sample(soil, beta)}) & = & \ell_1(\texttt{commun(soil)}) & = \\
\ell_1(\texttt{drive(beta, alpha)}) & = & \ell_1(\texttt{drive(beta, gamma)}) & = \\
\ell_1(\texttt{drive(gamma, alpha)}) & = & \ell_1(\texttt{drive(gamma, beta)}) & = \\
\ell_1(\texttt{at(alpha)}_p) & = & \ell_1(\texttt{at(beta)}_p) & = \\
\ell_1(\texttt{at(gamma)}_p) & & & = \{x_1^0, x_1^1, x_1^2, x_1^3\} \\
\ell_1(\texttt{avail(soil, alpha)}_p) & & & = \{x_1^0, x_1^1\} \\
\ell_1(\texttt{have(soil)}_p) & & & = \{x_1^1\} \\
\ell_1(\texttt{avail(soil, beta)}_p) & & & = \{x_1^2, x_1^3\}
\end{array}$$

This action layer allows several new actions, namely the `sample(soil, beta)` action, the `commun(soil)` action, and several `drive` actions. Each of the new actions is applicable in all possible worlds.

The level one effect layer is:

$$\begin{aligned}\mathcal{E}_1 = \mathcal{E}_0 \cup\ & \{\varphi_{00}(\texttt{sample(soil, beta)}), \varphi_{00}(\texttt{commun(soil)}),\\ & \varphi_{00}(\texttt{drive(beta, alpha)}), \varphi_{00}(\texttt{drive(beta, gamma)}),\\ & \varphi_{00}(\texttt{drive(gamma, alpha)}), \varphi_{00}(\texttt{drive(gamma, beta)}),\\ & \varphi_{00}(\texttt{have(soil)}_p), \varphi_{00}(\texttt{at(beta)}_p),\\ & \varphi_{00}(\texttt{at(gamma)}_p)\}\end{aligned}$$

$$\begin{aligned}
&\ell_1(\varphi_{00}(\texttt{drive(alpha, beta)})) = \ell_1(\varphi_{00}(\texttt{drive(alpha, gamma)})) =\\
&\ell_1(\varphi_{00}(\texttt{drive(beta, alpha)})) = \ell_1(\varphi_{00}(\texttt{drive(beta, gamma)})) =\\
&\ell_1(\varphi_{00}(\texttt{drive(gamma, alpha)})) = \ell_1(\varphi_{00}(\texttt{drive(gamma, beta)})) =\\
&\ell_1(\varphi_{00}(\texttt{at(beta)}_p)) = \ell_1(\varphi_{00}(\texttt{at(gamma)}_p)) =\\
&\ell_1(\varphi_{00}(\texttt{at(alpha)}_p)) = \{x_2^0, x_2^1, x_2^2, x_2^3\}\\
&\ell_1(\varphi_{00}(\texttt{sample(soil, alpha)})) = \ell_1(\varphi_{00}(\texttt{commun(soil)})) =\\
&\ell_1(\varphi_{00}(\texttt{have(soil)}_p)) = \{x_2^1\}\\
&\ell_1(\varphi_{00}(\texttt{avail(soil, alpha)}_p)) = \{x_2^0, x_2^1\}\\
&\ell_1(\varphi_{10}(\texttt{sample(soil, alpha)})) = \{x_2^0\}\\
&\ell_1(\varphi_{00}(\texttt{sample(soil, beta)})) = \ell_1(\varphi_{00}(\texttt{avail(soil, beta)}_p)) = \{x_2^2, x_2^3\}
\end{aligned}$$

Again, the particles are simulated by sampling each outcome for each action. The same particles sample the same outcomes of the `sample(soil, alpha)` action. Every particle samples the outcome 0 of the `sample(soil, beta)` action, but only two particles label the effect because its antecedent is only supported by those particles. The `commun(soil)` action has every particle sample outcome 0 and the effect is only supported in one particle.

The level two proposition layer is as follows:

$$\mathcal{P}_2 = \mathcal{P}_1 \cup \{\texttt{comm(soil)}\}$$

$$\begin{aligned}
\ell_2(\texttt{at(alpha)}) &= \ell_2(\texttt{at(beta)}) = \\
\ell_2(\texttt{at(gamma)}) &= \{x_2^0, x_2^1, x_2^2, x_2^3\} \\
\ell_2(\texttt{comm(soil)}) &= \{x_2^1\} \\
\ell_2(\texttt{have(soil)}) &= \{x_2^1, x_2^2, x_2^3\} \\
\ell_2(\texttt{avail(soil, alpha)}) &= \{x_2^0, x_2^1\} \\
\ell_2(\texttt{avail(soil, beta)}) &= \{x_2^2, x_2^3\}
\end{aligned}$$

The `have(soil)` proposition is now reachable by three particles, and the goal proposition `comm(soil)` is reachable by one particle (indicating the probability of goal satisfaction is 1/4). Notice also that the size of the effect layer is not necessarily monotonically increasing; some outcomes are not sampled at some levels, and their effects do not appear. However, the proposition and action layers are guaranteed to monotonically increase.

Continuing to the level two action layer:

$$\mathcal{A}_2 = \mathcal{A}_1 \cup \{\texttt{comm(soil)}_p\}$$

$$\begin{aligned}
\ell_2(\texttt{sample(soil, alpha)}) &= \ell_2(\texttt{drive(alpha, beta)}) &= \\
\ell_2(\texttt{drive(alpha, gamma)}) &= \ell_2(\texttt{at(alpha)}_p) &= \\
\ell_2(\texttt{sample(soil, beta)}) &= \ell_2(\texttt{drive(beta, alpha)}) &= \\
\ell_2(\texttt{drive(beta, gamma)}) &= \ell_2(\texttt{drive(gamma, alpha)}) &= \\
\ell_2(\texttt{drive(gamma, beta)}) &= \ell_2(\texttt{at(alpha)}_p) &= \\
\ell_2(\texttt{at(beta)}_p) &= \ell_2(\texttt{at(gamma)}_p) &= \\
\ell_2(\texttt{commun(soil)}) & &= \{x_2^0, x_2^1, x_2^2, x_2^3\}
\end{aligned}$$

$$
\begin{aligned}
\ell_2(\texttt{avail(soil, alpha)}_p) &= \{x_2^0, x_2^1\}\\
\ell_2(\texttt{have(soil)}_p) &= \{x_2^1, x_2^2, x_2^3\}\\
\ell_2(\texttt{avail(soil, beta)}_p) &= \{x_2^2, x_2^3\}\\
\ell_2(\texttt{comm(soil)}_p) &= \{x_2^1\}
\end{aligned}
$$

This action layer is exactly the same as the previous action layer, but includes a new persistence for `comm(soil)`.

The level two effect layer is:

$$
\begin{aligned}
\mathcal{E}_2 = \mathcal{E}_1 \quad & \cup \quad \{\varphi_{10}(\texttt{sample(soil, beta)}), \varphi_{10}(\texttt{commun(soil)}, \varphi_{00}(\texttt{comm(soil)}_p)\}\\
& \setminus \quad \{\varphi_{10}(\texttt{sample(soil, alpha)}\}
\end{aligned}
$$

$$
\begin{array}{lll}
\ell_2(\varphi_{00}(\texttt{drive(alpha, beta)})) & = \ell_2(\varphi_{00}(\texttt{drive(alpha, gamma)})) & = \\
\ell_2(\varphi_{00}(\texttt{drive(beta, alpha)})) & = \ell_2(\varphi_{00}(\texttt{drive(beta, gamma)})) & = \\
\ell_2(\varphi_{00}(\texttt{drive(gamma, alpha)})) & = \ell_2(\varphi_{00}(\texttt{drive(gamma, beta)})) & = \\
\ell_2(\varphi_{00}(\texttt{at(beta)}_p)) & = \ell_2(\varphi_{00}(\texttt{at(gamma)}_p)) & = \\
\ell_2(\varphi_{00}(\texttt{at(alpha)}_p)) & & = \{x_3^0, x_3^1, x_3^2, x_3^3\}\\
\ell_2(\varphi_{00}(\texttt{commun(soil)})) & & = \{x_3^2, x_3^3\}\\
\ell_2(\varphi_{00}(\texttt{have(soil)}_p)) & & = \{x_3^1, x_3^2, x_3^3\}\\
\ell_2(\varphi_{00}(\texttt{comm(soil)}_p)) & = \ell_2(\varphi_{10}(\texttt{commun(soil)})) & = \{x_3^1\}\\
\ell_2(\varphi_{00}(\texttt{sample(soil, alpha)})) & = \ell_2(\varphi_{00}(\texttt{avail(soil, alpha)}_p)) & = \{x_3^0, x_3^1\}\\
\ell_2(\varphi_{00}(\texttt{sample(soil, beta)})) & & = \{x_3^2\}\\
\ell_2(\varphi_{10}(\texttt{sample(soil, beta)})) & & = \{x_3^3\}\\
\ell_2(\varphi_{00}(\texttt{avail(soil, beta)}_p)) & & = \{x_3^2, x_3^3\}
\end{array}
$$

In this layer, the conditional effect of `commun(soil)` is sampled by two new particles, which have support from the antecedent `have(soil)`.

The level three proposition layer is as follows:

$$\mathcal{P}_3 = \mathcal{P}_2$$

$$\begin{aligned}
\ell_3(\texttt{at(alpha)}) &= \ell_3(\texttt{at(beta)}) &=& \\
\ell_3(\texttt{at(gamma)}) &= \ell_3(\texttt{have(soil)}) &=& \{x_3^0, x_3^1, x_3^2, x_3^3\} \\
\ell_3(\texttt{comm(soil)}) & &=& \{x_3^1, x_3^2, x_3^3\} \\
\ell_3(\texttt{avail(soil, alpha)}) & &=& \{x_3^0, x_3^1\} \\
\ell_3(\texttt{avail(soil, beta)}) & &=& \{x_3^2, x_3^3\}
\end{aligned}$$

The `comm(soil)` proposition is now reachable by 3/4 of the particles, meaning it is reachable with probability 0.75. Since the goal threshold is 0.5 probability, a relaxed plan supporting the 3 particles labeling the goal proposition will reflect the cost to achieve the goal.

Termination: $\mathcal{M}cLUG$ construction continues until a proposition layer supports the goal with probability no less than τ. I assess the probability of the goal at level k by finding the set of particles where the goal is supported and taking the ratio of its size with N. Formally,

$$P(G|X_k) \approx \frac{|\bigcap_{p \in G} \ell_k(p)|}{N}$$

I also define level off for the $\mathcal{M}cLUG$ as the condition when every proposition in a proposition layer is labeled with the same number of particles as in the previous level. If level off is reached without $p(G|X_k) \geq \tau$, then we set the heuristic value of the source belief state to ∞.

4.2. Heuristics. I just defined how to terminate construction of the $\mathcal{M}cLUG$ and we can use the resulting depth k as a measure of the number of steps needed to achieve the goal with probability no less than τ. This heuristic is similar to the max heuristic defined for the *LUG*.

As has been shown in non-deterministic and classical planning, relaxed plan heuristics are often much more effective, despite being inadmissible. Since I am already approximating the possible world distribution of the planning graph and losing admissibility, I will use relaxed plans as the heuristic. The relaxed plan extraction is almost identical to the relaxed plan extraction in the *LUG*. The extraction is very fast because it makes use of the symbolic representation to obtain a relaxed plan for all particles at once, rather than each individually and aggregating them. The intuition behind the relaxed plan is that I know which particles support the goal propositions and which paths the particles took through the $\mathcal{M}$*cLUG*, so I can pick actions, labeled with these particles, that support the goal.

Often there are many choices for supporting a subgoal in a set of particles. Consider a subgoal g that must be supported in a set of particles $\{x_k^1, x_k^2, x_k^3\}$ and is supported by effect φ in particles x_k^1 and x_k^2, φ' in particles x_k^2 and x_k^3, and φ'' in x_k^2. Choosing support in the wrong order may lead me to include more actions than needed, especially if the effects are of different actions. This problem is actually a set cover, which I solve greedily. For example, until the set of particles for g is covered, I select supporting effects based on the number of new particles they cover (except for proposition persistence actions, which I prefer over all others). The number of particles an effect can support is proportional to the probability with which the effect supports the proposition. Say I first pick φ because it covers two new particles, then φ' can cover one new particle, and φ'' covers no new particles. I finish the cover by selecting φ' for particle x_k^3. Notice that even though φ' can support two particles I use it to support one. When the relaxed plan extraction subgoals to support φ' it only supports it in particle x_k^3 to avoid “bloating” the relaxed plan.

5. Empirical Evaluation

I externally evaluate $POND$ using the $\mathcal{M}c$*LUG* relaxed plan heuristic by comparing with the leading approach to probabilistic conformant planning, CPplan (Hyafil and Bacchus, 2003, 2004). I also internally evaluate my approach by adjusting the number of particles N used in each $\mathcal{M}c$*LUG*. I refrain from comparing with POMDP solvers, as did (Hyafil and Bacchus, 2004), because they were shown to be effective only on problems with very small state spaces (e.g., slippery gripper and sandcastle-67) and I care about problems with large state spaces. My approach does only slightly better than CPplan on the small state space problems and I doubt we are superior to the POMDP solvers on these problems.

I use four test domains for the evaluation: logistics, grid, slippery gripper, and sandcastle-67. In the test setup, I used a 2.66 GHz P4 Linux machine with 1GB of memory, with a timeout of 20 minutes for each problem. I note that CPplan performs marginally worse than previously reported because my machine has one third the memory of the machine that Hyafil and Bacchus (2004) used for their experiments.

CPplan is an optimal bounded length planner that uses a CSP solver for probabilistic conformant planning. Part of the reason CPplan works so well is its efficient caching scheme that re-uses optimal plan suffixes to prune possible solutions. In comparison, my work computes a relaxation of plan suffixes to heuristically rank partial solutions. CPplan finds the optimal probability of goal satisfaction for a given plan length (an NP^{PP}-complete problem (Littman *et al.*, 1998)), but $POND$, like Buridan (Kushmerick *et al.*, 1994), finds plans that satisfy the goal with probability no less than τ (an undecidable problem (Madani *et al.*, 1999)). CPplan could be used to find an optimal length plan that exceeds τ by iterating over increasing plan lengths (similar to BlackBox (Kautz *et al.*, 1996)).

To compare with CPplan, I run CPplan on a problem for each plan length until it exceeds the time or memory limit. I record the probability that CPplan satisfies the goal for each plan length. I then give $POND$ a series of problems with increasing values for τ (which match the values found by CPplan). If $POND$ can solve the problem for all values of τ solved by CPplan, then I increase τ by fixed increments thereafter. I ran $POND$ five times on each problem and present the average run time, plan length, and expanded search nodes. Comparing the planners in this fashion allows me to compare the plan lengths found by $POND$ to the optimal plan lengths found by CPplan for the same value of τ. $POND$ often finds plans that exceed τ and include more actions, whereas CPplan meets τ with the optimal number of actions. Nevertheless, I feel the comparison is fair and illustrates the pro/cons of an optimal planner with respect to a heuristic planner.

Logistics: The logistics domain has the standard logistics actions of un/loading, driving, and flying, but adds uncertainty. Hyafil and Bacchus (2004) enriched the domain developed by Brafman and Hoffmann (2004) to not only include initial state uncertainty, but also action uncertainty. In each problem there are some number of packages whose probability of initial location is uniformly distributed over some locations and un/loading is only probabilistically successful. Plans require several loads and unloads for a single package at several locations, making a relatively simple deterministic problem a very difficult stochastic problem. I compare on three problems p2-2-2, p4-2-2, and p2-2-4, where each problem is indexed by the number of possible initial locations for a package, the number of cities, and the number of packages. See (Hyafil and Bacchus, 2004) for more details.

The plots in Figures 61 and 62 compare the total run time in seconds (left) and the plan lengths (right) of $POND$ using 16/32/64/128 particles in the $\mathcal{M}cLUG$ versus CPplan. In this domain I also use helpful actions from the relaxed plan (Hoffmann and Nebel, 2001). CPplan is able to at best find solutions where $\tau \leq 0.26$ in p2-2-2, $\tau \leq 0.09$ in p4-2-2, and $\tau \leq 0.03$ in p2-2-4. In most cases $POND$ is able to find plans much faster than CPplan for the problems they both solve. It is more interesting that $POND$ is able to solve problems for *much larger* values of

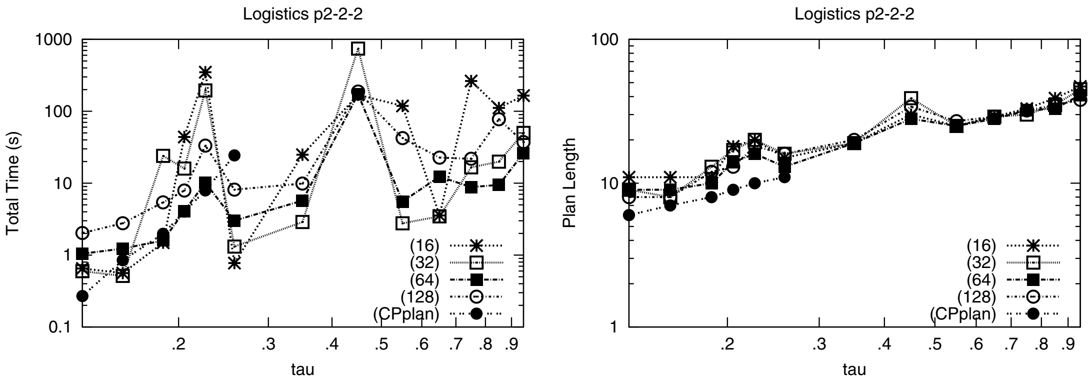


Figure 61. Run times (s) and Plan lengths vs. τ (log scale) for Logistics p2-2-2

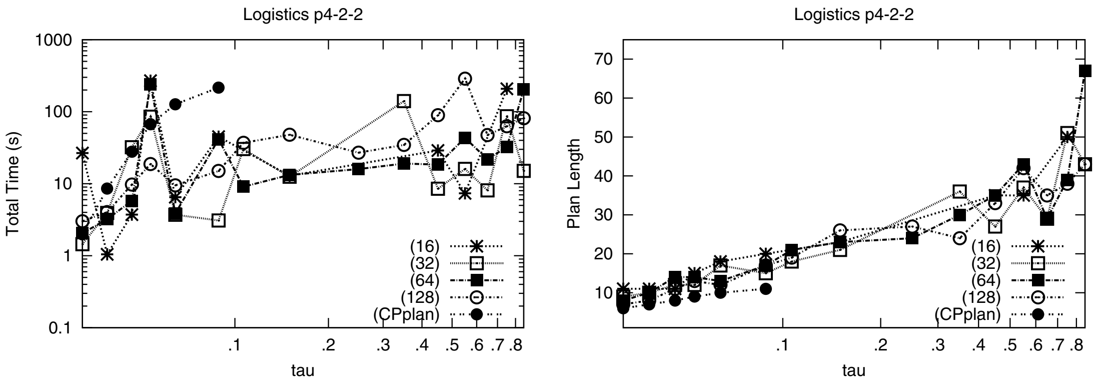


Figure 62. Run times (s) and Plan lengths vs. τ (log scale) for Logistics p4-2-2

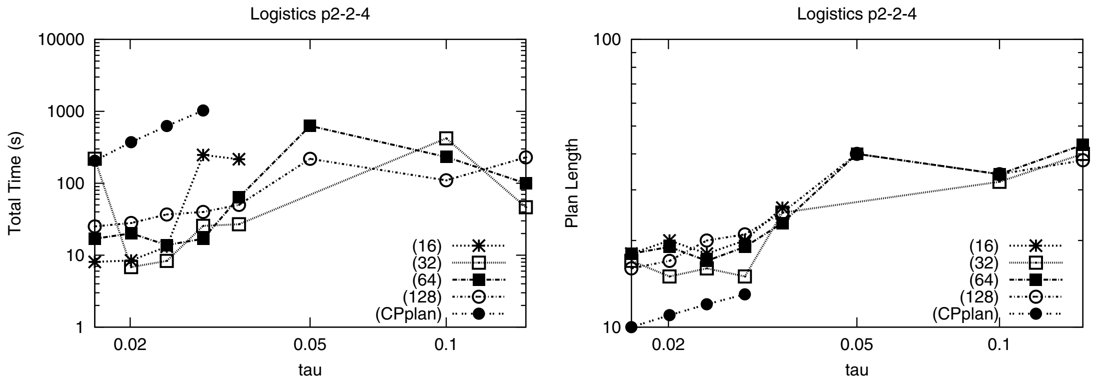


Figure 63. Run times (s) and Plan lengths vs. τ (log scale) for Logistics p2-2-4

τ. $POND$ finds solutions where $\tau \leq 0.95$ in p2-2-2, $\tau \leq 0.85$ in p4-2-2, and $\tau \leq 0.15$ in p2-2-4, which is respectively 3.7, 9.6, 5.2 times the maximum values of τ solved by CPplan. In terms of

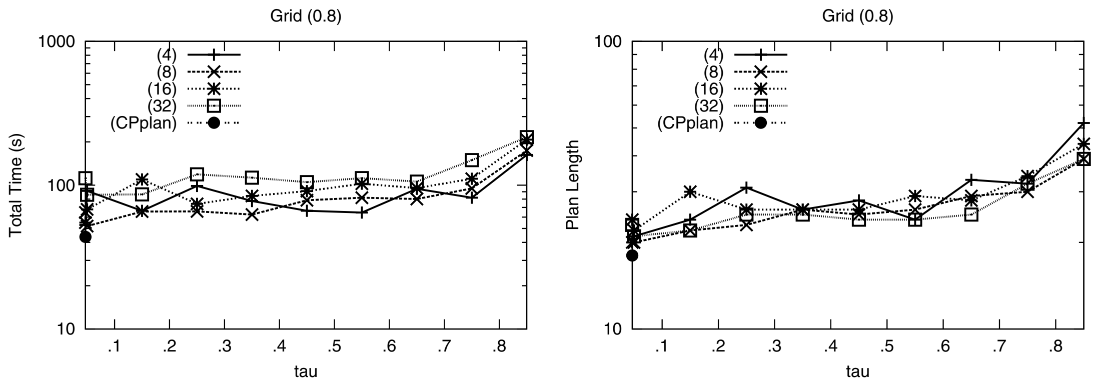


Figure 64. Run times (s) and Plan lengths vs. τ for Grid-0.8

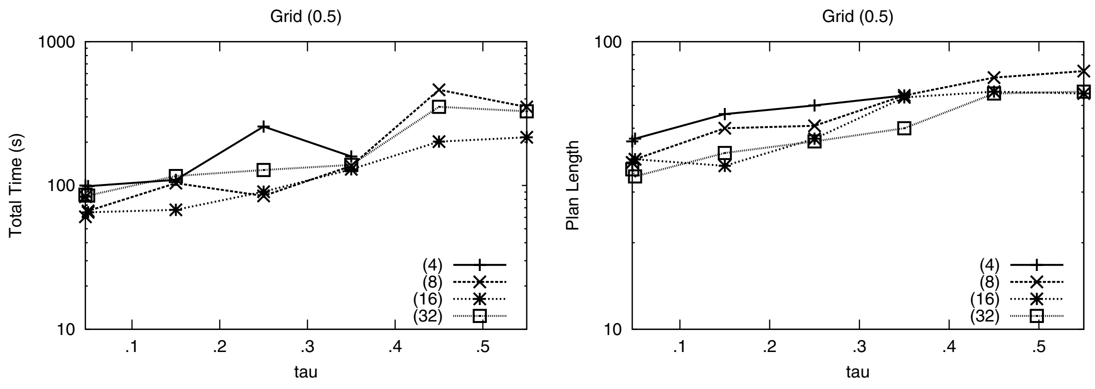


Figure 65. Run times (s) and Plan lengths vs. τ for Grid-0.5

plan quality, the average increase in plan length for the problems we both solved was 4.6 actions in p2-2-2, 4.2 actions in p4-2-2, and 6.4 actions in p2-2-4. I believe that some of the spikes in run times as τ increases are due to the $\mathcal{M}cLUG$ coming very close to τ in a layer, but extending an extra layer to exceed τ. When the $\mathcal{M}cLUG$ extends the extra layer, the relaxed plan will likely contain more actions and increase the heuristic estimate. This could be leading search away from easy solutions that will exceed τ.

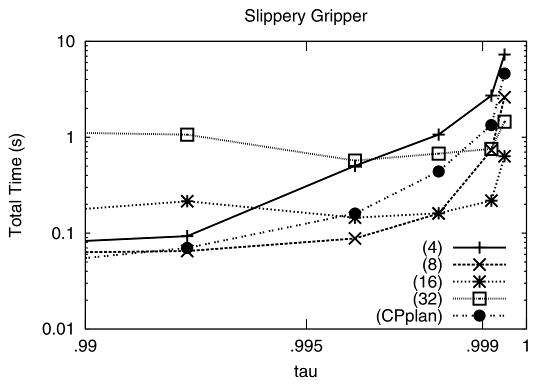
The plot of plan lengths gives some intuition for why CPplan has trouble finding plans for greater values of τ. The plan lengths for the larger values of τ approach 40-50 actions and CPplan is limited to plans of around 10-15 actions. For $POND$ we notice that plan length and total time scale roughly linearly as τ increases. We can see that the $\mathcal{M}c$*LUG* relaxed plan heuristic directs search very well.

I would also like to point out some differences in how $POND$ performs when the number of particles changes. As τ increases, using more particles makes the search more consistent (i.e., fluctuation in terms of run time and plan length is minimized). Total time generally increases as the number of particles increases because the heuristic is costlier.

Grid: The Grid domain, as described by (Hyafil and Bacchus, 2004), is a 10x10 grid where a robot can move one of four directions to adjacent grid points. The robot has imperfect effectors and moves in the intended direction with high probability (0.8), and in one of the two perpendicular directions with a low probability (0.1). As the robot moves, its belief state grows and it becomes difficult to localize itself. The goal is to reach the upper corner of the grid. The initial belief state is a single state where the robot is at a known grid point. I test on the most difficult instance where the robot starts in the lower opposite corner.

Figure 64 shows total run time and plan lengths for the problem. I notice that CPplan can solve the problem for only the smallest values of τ, whereas $POND$ scales much better. For the single problem we both solve, $POND$ is on average finding solutions with 4.75 more actions. Again, $POND$ scales roughly linearly because the $\mathcal{M}c$*LUG* heuristic is very informed. In this problem, $POND$ is able to do very well with only 4-8 particles, leading me to believe that there are only a few very important regions of the distribution over possible worlds and the $\mathcal{M}c$*LUG* actually captures them.

Doing so well with only a few particles made me question whether the $\mathcal{M}c$*LUG* is really needed. As a sanity check, I show results for a variation of the grid problem in Figure 65. This problem defines the probability that the robot moves in the intended direction to 0.5 and to 0.25 for the adjacent directions. The result is that as the robot moves, the belief state will be much less



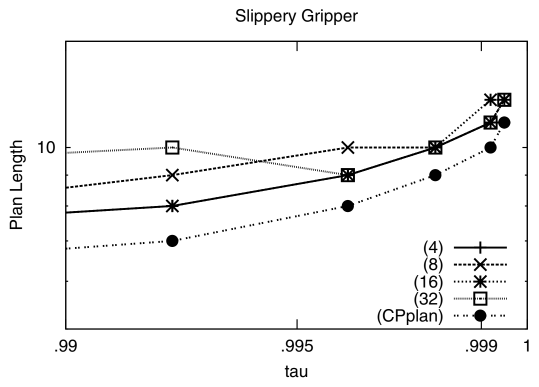


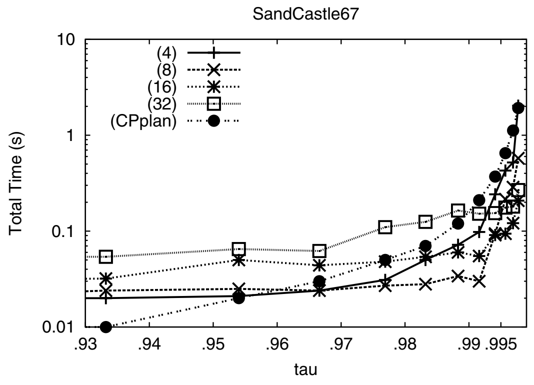
Figure 66. Run times (s) and Plan lengths vs. τ for Slippery Gripper.

peaked and harder to capture with few particles. I see that my doubts are quieted by the results. More particles are required to get good quality plans and make search more effective.

Slippery Gripper: Slippery Gripper is a well known problem that was originally presented by Kushmerick *et al.* (1994). There are four probabilistic actions that clean and dry a gripper and paint and pick-up a block. The goal is to paint the block and hold it with a clean gripper. Many of the lower values of τ find very short plans and take very little run time, so I focus on the higher values of τ where I see interesting scaling behavior.

Figure 67 shows the total time and plan length results for this problem in the two left-most plots. For short plans, CPplan is faster because the $\mathcal{M}cLUG$ has some additional overhead, but as τ increases and plans have to be longer the $\mathcal{M}cLUG$ proves useful. Using 8 particles, $POND$ is able to find solutions faster than CPplan in the problems where $\tau > .99$. Using more particles, $POND$ is able to find solutions faster for most problems where $\tau \geq .998$. In terms of plan quality, the $POND$ solutions are usually 1-2 extra actions with 3 extra actions in the worst case.

SandCastle-67: SandCastle-67 is another well known probabilistic planning problem, presented by Majercik and Littman (1998). The task is to build a sand castle with high probability by using two actions: erect-castle and dig-moat. Having a moat improves the probability of successfully erecting



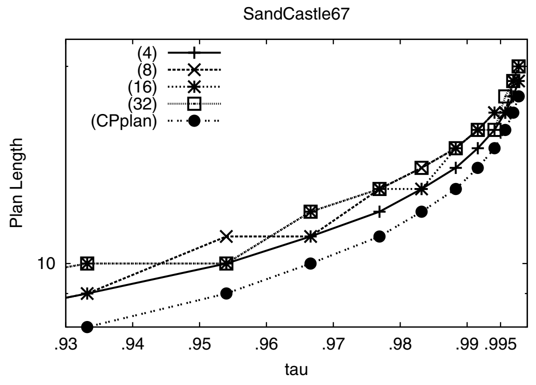


Figure 67. Run times (s) and Plan lengths vs. τ for SandCastle-67.

a castle, but erecting a castle may destroy the moat. Again, scaling behavior is only interesting for high values of τ.

In the two right-most plots for run time and plan length in Figure 67, I see that the run time for CPplan has an exponential growth with τ, whereas $POND$ scales roughly linearly. As τ increases, $POND$ is eventually able to outperform CPplan. In terms of plan quality, $POND$ usually find plans with 1-3 extra actions.

Discussion: In comparison with CPplan, the major difference with my heuristic approach is the way that plan suffixes are evaluated. CPplan must exactly compute plan suffixes to prune solutions, whereas I estimate plan suffixes. It turns out that the estimates require $POND$ to evaluate very few possible plans. As plans become longer, it is more difficult for CPplan to exactly evaluate plan suffixes because there are so many and they are large.

By adjusting the number of particles within $POND$ I have a couple of general observations. As can be expected, when belief states or distributions over possible worlds are fairly non-peaked distributions, using more particles guides search better. However, without understanding the domain, it is difficult to pick the right number of particles. Fortunately, the number of particles is an easily tunable parameter that cleanly adjusts the all-too-common cost/effectiveness tradeoff in heuristics. It would be interesting to use techniques from particle filter research to automatically determine the right number of particles during search.

Overall, my method is very effective in the probabilistic conformant planning problems I evaluated, with the only drawback being longer plans in some cases. To compensate, I believe it should be reasonably straight-forward to post-process the plans to cut cost by removing actions. Nevertheless, it is a valuable lesson to see the size of problems that can be solved by relaxing our grip on finding optimal plans.

6. Related Work

Buridan (Kushmerick *et al.*, 1994) was one of the first planning algorithms to solve probabilistic conformant planning. Buridan is a partial order casual link (POCL) planner that allows multiple supporters for an open condition, much like our relaxed plans in the $\mathcal{M}cLUG$. Unfortunately, Buridan does not scale very well because it lacks effective heuristics. Probapop (Onder *et al.*, 2006), which is built on top of Vhpop (Younes and Simmons, 2003), attempts to enhance Buridan by using heuristics. Probapop uses the classical planning graph heuristics implemented by Vhpop by translating every outcome of probabilistic actions to a deterministic action. In theory, POCL planners are a nice framework for probabilistic planning because it is easy to add actions to support a low probability condition without backtracking (as may be necessary in state based search). In reality, POCL can be hard to work with because it is often difficult to assess the probability of a partially ordered plan.

Partially observable Markov decision process (POMDP) algorithms, such as (Cassandra *et al.*, 1997) to name one, are also able to solve probabilistic conformant planning. The work on CPplan (Hyafil and Bacchus, 2003, 2004) makes extensive comparisons with the mentioned POMDP algorithm and shows it is inferior for solving probabilistic conformant planning problems with large state spaces (like logistics and grid). This disparity may be partly due to the fact that the POMDP algorithms solve a slightly different problem by finding plans for all possible initial belief states. CPplan also compares with MaxPlan (Majercik and Littman, 1998), showing it too is inferior for several problems. MaxPlan is similar to CPplan, in that it encodes probabilistic conformant planning as a bounded length planning problem using a variant of satisfiability. The main difference is in the way they cache plan suffixes.

More closely related to our approach is the work on the CFF planner (Brafman and Hoffmann, 2004). In this work the focus is to use planning graphs to derive relaxed plan heuristics for non-deterministic conformant planning. CFF encodes a relaxed planning graph as a tractable satisfiability instance and tries to prove a relaxed plan exists for every model of a belief state. The work on the *LUG* directly propagates constraints about possible worlds on a planning graph and extracts a relaxed plan.

The Prottle planner (Little *et al.*, 2005) uses a variation of temporal planning graphs for fully-observable probabilistic temporal planning. In their planning graph they explicitly reason about actions with probabilistic actions by adding an outcome layer and defining a cost propagation procedure. The authors do not extract relaxed plans, nor reason about possible worlds.

PGraphPlan (Blum and Langford, 1999) and CGP (Smith and Weld, 1998) are two planners that use generalizations of GraphPlan (Blum and Furst, 1995) for planning under uncertainty. PGraphPlan and its sister algorithm TGraphPlan, are used for fully-observable probabilistic planning (similar to Markov decision processes). The key idea in PGraphPlan is to forward chain in the planning graph, using dynamic programming, to find an optimal probabilistic plan for a given finite horizon. Alternatively, TGraphPlan greedily back-chains in the planning graph to find a solution that satisfies the goal, without guaranteeing optimality. CGP solves non-observable (conformant) non-deterministic planning problems.

RTDP (Barto *et al.*, 1995) is a popular algorithm, used in many recent works (Mausam and Weld, 2005; Little *et al.*, 2005; Bonet and Geffner, 2003), that also uses Monte Carlo techniques. RTDP samples a plan suffix to evaluate, whereas I estimate the plan suffix with a relaxed plan. Because I am reasoning about non-observable problems I sample several plan suffixes and aggregate them to reflect that we are planning in belief space.

7. Conclusion

In this chapter, I have presented an approach called $\mathcal{M}cLUG$ to integrate sequential Monte Carlo approaches into heuristic computation on planning graphs. The $\mathcal{M}cLUG$ enables quick computation of effective heuristics for conformant probabilistic planning. By using the heuristics, $POND$ is able to far out-scale the current best approach to conformant probabilistic planning. At a broader level, this work shows one fruitful way of exploiting the recent success in deterministic planning to scale stochastic planners.

CHAPTER 7

STATE AGNOSTIC PLANNING GRAPHS

Previous chapters describe how to use planning graphs in a number of scenarios, all of which assume a new planning graph (or set of planning graphs) is built for every search node. This chapter takes a more general view of labeled planning graphs to describe how they can represent the entire set of planning graphs for every search node. I explore this idea, called a state agnostic planning graph, in classical, non-deterministic, and stochastic planning with uniform cost actions. I consider non-deterministic and probabilistic conformant planning and non-deterministic conditional planning.

1. Introduction

Heuristics derived from planning graphs (Blum and Furst, 1995) are widespread in planning (Gerevini *et al.*, 2003; Hoffmann and Nebel, 2001; Bonet and Geffner, 1999; Younes and Simmons, 2003; Nguyen *et al.*, 2002). In many cases, heuristics are derived from a set of planning graphs. In classical planning, progression planners compute a planning graph for every search state. (The same situation arises in planning under uncertainty when calculating the heuristic for a belief state, as we have seen in previous chapters.) A set of planning graphs for related states can be highly redundant. That is, any two planning graphs often overlap significantly. As an extreme example, the planning graph for a child state is a sub-graph of the planning graph of the parent, left-shifted by one step (Zimmerman and Kambhampati, 2005). Computing a set of planning graphs by enumerating its members is, therefore, inherently redundant.

Consider progression planning in a variation of the classical rovers domain, described in Figure 68. There are two locations, `alpha` and `gamma`, and an `image` is available at `gamma`. The goal is to achieve `(comm image)`. There are four actions `drive(alpha, gamma)`, `drive(gamma, alpha)`, `sample(image, gamma)`, and `commun(image)`. The rover can use the plan: `drive(alpha, gamma)`, `sample(image, gamma)`, `commun(image)` to achieve the goal. The sequence of states, where each state will be evaluated by the heuristic, is:

$s_I = \{$`at(alpha), avail(image, gamma)`$\}$

$s_2 = \{$`at(gamma), avail(image, gamma)`$\}$

$s_{10} = \{$`at(gamma), have(image), avail(image, gamma)`$\}$

$s_{14} = \{$`at(gamma), have(image), commun(image), avail(image, gamma)`$\}$

Notice that $s_2 \subset s_{10} \subset s_{14}$, meaning that the planning graphs for each state will have initial proposition layers where $\mathcal{P}_0(s_2) \subset \mathcal{P}_0(s_{10}) \subset \mathcal{P}_0(s_{14})$. Further, many of the same actions will appear in the first action layer of the planning graph for each state. It is possible to capture this set of planning graphs by computing a *LUG* for the hypothetical belief state $b = \{s_2, s_{10}, s_{14}\}$. We can use the *LUG* to represent this set of planning graphs, but how do we compute heuristics for individual states?

The answer, developed in this chapter, is an elegant generalization of the planning graph called the State Agnostic Graph (SAG), that extends the labeling technique introduced in the *LUG*. In this chapter, I take a more general perspective on the labeling technique employed, described here as SAG, and apply this perspective to further boost the performance of $POND$, employing *LUG*-based heuristics.

It is common to view the planning graph as exactly solving a single-source shortest path problem, for a relaxed planning problem. The levels of the graph efficiently represent a breadth-first sweep from the single source. In the context of progression planning, the planner will end up calculating a heuristic for many different sources. Iterating a single-source algorithm over each source (building a planning graph per search node) is a naive solution to the all-pairs shortest path problem. I develop a generalization, SAG, under this intuition: directly solving the all-pairs shortest path problem is more efficient than iterating a single source algorithm.

This intuition falls short in most planning problems, because the majority of states are unreachable. Reasoning about such states is useless, so instead, I develop the SAG as a solution to the multi-source shortest path problem. More precisely, the SAG is a representation of a set of planning graphs. Its exact form depends upon the kind of planning graph being generalized (mutexes, cost, time, . . .). The main insight to the technique is to represent the propagation rules of the planning graph and a set of sources as boolean functions. Composing these functions via boolean algebra yields an approach for building the set of planning graphs without explicitly enumerating its elements. Manipulating formulas exploits redundant sub-structure, and boosts empirical performance, in spite of the fact that this technique does not alter theoretical worst-case complexity. While the representation is in terms of boolean functions, I describe the label semantics in terms of sets (models of the boolean functions).

I start by introducing the SAG technique, for the purpose of exposition, in the simpler realm of classical planning. I restrict attention to planning graphs without mutexes: relaxed planning graphs. I generalize this planning graph to the State Agnostic Graph (SAG). I extend the discussion to belief-space planning, generalizing the *LUG* to its state agnostic version, and generalizing the $\mathcal{M}c$*LUG* to its state agnostic version. I then demonstrate two formalizations, $SLUG$ and $\mathcal{C}SSAG$, which drastically improve theoretical complexity: I use these results as the basis for heuristics in $POND$. I also explore a generalization of relaxed plan heuristics that follows directly from the SAG, namely, the state agnostic relaxed plan, which captures the relaxed plan for every state. From there, I consider several strategies for reducing irrelevant heuristic computations. The experimental evaluation begins by comparing these strategies. I take the best among these to conduct an external comparison with state of the art belief space planners. I demonstrate that POND is competitive with the state of the art in both conformant and conditional non-deterministic planning and conformant probabilistic planning (the next chapter covers conditional probabilistic planning). Before

```
(define (domain rovers_classical)
  (:requirements :strips :typing)
  (:types location data)
  (:predicates
           (at ?x - location)
           (avail ?d - data ?x - location)
           (comm ?d - data)
           (have ?d - data))
  (:action drive
   :parameters (?x ?y - location)
   :precondition (at ?x)
   :effect (and (at ?y) (not (at ?x))))
  (:action commun
   :parameters (?d - data)
   :precondition (have ?d)
   :effect (comm ?d))
  (:action sample
   :parameters (?d - data ?x - location)
   :precondition (and (at ?x) (avail ?d ?x))
   :effect (have ?d))
)
```

```
(define (problem rovers_classical1)
  (:domain rovers_classical)
  (:objects
      image - data
      alpha gamma - location)
  (:init (at alpha)
         (avail image gamma))
  (:goal (and (comm image)))
)
```

Figure 68. PDDL description of classical planning formulation of the rover problem.

concluding, I provide some insight into the connections to other work.

2. Classical Planning

I first present the SAG in a classical setting by describing how it can represent the classical planning graphs discussed in Chapter 2, as well as how it is used to compute a simple level heuristic. Recall that a planning graph, $PG(s)$, built for a single source s, satisfies:

1. If p holds in s then $p \in \mathcal{P}_0(s)$

2. For any k such that $p \in \mathcal{P}_k$ for every $p \in \rho_e(a)$, then $a \in \mathcal{A}_k$

3. For any k such that $a \in \mathcal{A}_k$, then $p \in \mathcal{P}_{k+1}$ for all $p \in \varepsilon^+(a)$

A proposition or action is reachable from a state s after at least k actions if it appears in level k of the planning graph built from state s. The level heuristic is based on the extension of this idea to a set of propositions: the level of a set of propositions is the least level k where each proposition appears.

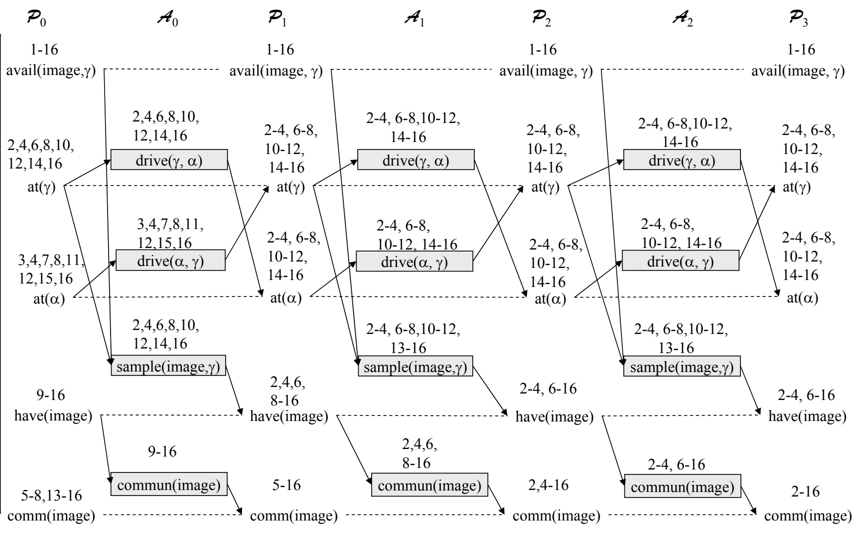
I generalize the planning graph to the State Agnostic Planning Graph (SAG), by permitting multiple source states. I associate every vertex of the graph with a label ℓ; each label tracks the set of sources reaching the associated graph vertex. These labels are represented as boolean functions of the domain propositions to describe sets. Intuitively, a source s reaches a graph vertex x if it is a subset of the label $s \subset \ell_k(x)$.

A State Agnostic Planning Graph is a labeled planning graph, with a label ℓ function assigning states to vertices. The graph $SAG(\mathcal{S})$ is constructed from its scope $\mathcal{S}$, a set of sources. For all $s \in \mathcal{S}$, the following holds:

1. If $p \in s$, then $s \in \ell_0(p)$ and $p \in \mathcal{P}_0$
2. If $s \in \ell_k(p)$ for every $p \in \rho_e(a)$, then:
   (a) $s \in \ell_k(a)$
   (b) $a \in \mathcal{A}_k$
3. If $p \in \mathit{eff}^+(a)$ and $s \in \ell_k(a)$, then $s \in \ell_{k+1}(p)$ and $p \in \mathcal{P}_{k+1}$

Consider building a SAG, in Figure 69, for the entire set of states $\mathcal{S} = S$ for the problem described in Figure 68 (omitting the static proposition `avail(image, gamma)` from state descriptions). There are a total of 16 states (4 of which are unreachable) listed in the table in Figure 69. The figure denotes the labels with sets of integers, which correspond to state indices in the table. The initial proposition layer contains every proposition, labeled with the states in which it holds. Label propagation is identical to the *LUG*: actions are labeled with the intersection of their precondition labels, and propositions are labeled with the union of their supporters. By the third level, the goal proposition `comm(soil)` is labeled with all but one state, and extending the SAG

S	have	comm	at(α)	at(γ)
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1
5	0	1	0	0
6	0	1	0	1
7	0	1	1	0
8	0	1	1	1
9	1	0	0	0
10	1	0	0	1
11	1	0	1	0
12	1	0	1	1
13	1	1	0	0
14	1	1	0	1
15	1	1	1	0
16	1	1	1	1



$\mathcal{P}_0$
$\mathcal{A}_0$
$\mathcal{P}_1$
$\mathcal{A}_1$
$\mathcal{P}_2$
$\mathcal{A}_2$
$\mathcal{P}_3$
1-16 avail(image,γ)
2,4,6,8,10, 12,14,16 at(γ)
3,4,7,8,11, 12,15,16 at(α)
9-16 have(image)
5-8,13-16 comm(image)
2,4,6,8,10, 12,14,16 drive(γ, α)
3,4,7,8,11, 12,15,16 drive(α, γ)
2,4,6,8,10, 12,14,16 sample(image,γ)
9-16 commun(image)
1-16 avail(image, γ)
2-4, 6-8, 10-12, 14-16 at(γ)
2-4, 6-8, 10-12, 14-16 at(α)
2,4,6, 8-16 have(image)
5-16 comm(image)
2-4, 6-8,10-12, 14-16 drive(γ, α)
2-4, 6-8, 10-12, 14-16 drive(α, γ)
2-4, 6-8,10-12, 13-16 sample(image,γ)
2,4,6, 8-16 commun(image)
1-16 avail(image, γ)
2-4, 6-8, 10-12, 14-16 at(γ)
2-4, 6-8, 10-12, 14-16 at(α)
2-4, 6-16 have(image)
2,4-16 comm(image)
2-4, 6-8,10-12, 14-16 drive(γ, α)
2-4, 6-8, 10-12, 14-16 drive(α, γ)
2-4, 6-8,10-12, 13-16 sample(image,γ)
2-4, 6-16 commun(image)
1-16 avail(image, γ)
2-4, 6-8, 10-12, 14-16 at(γ)
2-4, 6-8, 10-12, 14-16 at(α)
2-4, 6-16 have(image)
2-16 comm(image)

Figure 69. SAG for rover example.

an additional level will not change the labels. The SAG is said to level off at level 4, at which point it can be used to compute the heuristic for any state that is a subset of the scope $\mathcal{S}$.

The level heuristic for any state $s \in \mathcal{S}$ is the first level k where $s \in \bigcap_{p \in G} \ell_k(p)$, meaning the state can reach each goal proposition in k steps. To see the connection with the *LUG*, notice that the *LUG* built for a belief state b is using a scope $\mathcal{S} = b$, and the level heuristic for b is the first level k where $b = \bigcap_{p \in G} \ell_k(p)$. Using the SAG to compute the heuristic for a state involves *set membership* checks, whereas computing the heuristic for a belief state in the *LUG* uses a *set equivalence* check. Later I will show that the SAG can compute the heuristic for all belief states by using a *subset* check.

Sharing: Unlike the classical planning graph, the SAG can be reused across multiple search states. Consider a set of sources, $\mathcal{S}$. It is possible to extract a planning graph heuristic for each source, $s \in \mathcal{S}$, from $SAG(\mathcal{S})$, *without building any additional graphs*. This is done by evaluating labels. As an example, I presented a generalization of the level heuristic. The sharing of the SAG across invocations of the heuristic involves passing the graph as an argument to the heuristic function.

That is, the caller builds a graph for a set of anticipated search states, and passes that graph when obtaining the heuristic estimate for each search state in the set.

The graph, $SAG(\mathcal{S})$, is built once for a set of states, $\mathcal{S}$. For any $s \in \mathcal{S}$, the heuristic reuses the shared graph $SAG(\mathcal{S})$. For example, it is possible to compute the level heuristic for every state in the rover problem, by finding the first level where the label of `comm(soil)` contains the state. The states $\{s_5, s_6, s_7, s_8, s_{13}, s_{14}, s_{15}, s_{16}\}$ have a level heuristic of zero, the states $\{s_9, s_{10}, s_{11}, s_{12}\}$ have a heuristic of one, the states $\{s_2, s_4\}$ have a heuristic of two, and the state s_3 (the initial state) has a heuristic of three. It is possible to compute the heuristic values *a priori*, or on-demand during search. Later I will discuss various points along the continuum between computing all heuristic values before search and computing the heuristic at each search node when I describe relaxed plan heuristics.

The search tree for this rover problem has a total of five unique states $s_3, s_2, s_{10}, s_{11}, s_{14}$. By constructing a planning graph for each state, the total number of planning graph vertices (for propositions and actions) that must be allocated is 66. Constructing the equivalent SAG, while representing the planning graphs for extra states, requires 31 planning graph vertices. To answer the question of whether the SAG is more compact and efficient than building individual planning graphs, we must consider the size of labels and the associated manipulation costs. Representing labels as boolean functions (e.g., with BDDs) significantly reduces the size and cost of labels, making the SAG a better choice in some cases. We must also consider whether the SAG is used enough by the search: it may be too costly to build the SAG for all states if we only evaluate the heuristic for relatively few states. I will address these issues empirically in the evaluation section, but first consider the SAG for planning in non-deterministic belief state space.

```
(define (domain rovers_nd)
  (:requirements :strips :typing)
  (:types location data)
  (:predicates
          (at ?x - location)
          (avail ?d - data ?x - location)
          (comm ?d - data)
          (have ?d - data))
  (:action drive
   :parameters (?x ?y - location)
   :precondition (at ?x)
   :effect (and (at ?y) (not (at ?x))))
  (:action commun
   :parameters (?d - data)
   :precondition (and)
   :effect (when (have ?d) (comm ?d)))
  (:action sample
   :parameters (?d - data ?x - location)
   :precondition (at ?x)
   :effect (when (avail ?d ?x) (have ?d)))
)
```

```
(define (problem rovers_nd1)
  (:domain rovers_nd)
  (:objects
     image - data
     alpha gamma - location)
  (:init (at alpha)
        (avail image gamma)
        (oneof (have image)
                (not (have image))))
  (:goal (comm image))
)
```

Figure 70. PDDL description of non deterministic planning formulation of rover problem.

3. Non-Deterministic Planning

As previously mentioned, it is possible to apply the SAG to belief state space planning to compute the heuristic value of belief states. My presentation parallels the discussion in the classical setting: I revisit a planning graph-variant for belief-space planning, *LUG*, and generalize it to the state agnostic version (to represent a set of *LUG*). The worst-case complexity of the SAG technique is exponential in the worst-case complexity of the *LUG* method. That is, normally, there is no complexity-theoretic advantage to the technique. In the case of non-deterministic planning, I find an equivalent representation, $SLUG$ (see below), with the same worst-case complexity as *LUG*– an exponential reduction. The intuition is that the *LUG* operates on a set of states, and the naive generalization would operate on a set of sets of states. However, by taking the union of these sets of states, the $SLUG$ manages to reduce the complexity of the representation.

As before, I describe the actions in belief state space planning as having conditional effects.

I extend the running example to belief state space by reinterpreting its operators, as shown in Figure 70. The `commun(image)` and `sample(image, gamma)` actions have conditional effects. There is uncertainty as to whether `have(image)` is true, so the initial belief state is $b_0 = \{s_3, s_{11}\}$ (using the same state indices as the classical planning example above).

To reiterate, a Labeled Uncertainty Graph, *LUG*(b), is a labeled graph, built for the belief state b. Each label, ℓ, is a set world states, represented by a boolean function. For every $s \in b$, the following holds:

1. If $p \in s$, then $s \in \ell_0(p)$ and $p \in \mathcal{P}_0$
2. If $s \in \ell_k(p)$ for every $p \in \rho_e(a)$, then:
   (a) $s \in \ell_k(a)$
   (b) $a \in \mathcal{A}_k$
3. If $a \in \mathcal{A}_k$ and $s \in \ell_k(p) \cap \ell_k(a)$ for every $p \in \rho_{ij}(a)$, then:
   (a) $s \in \ell_k(\varphi_{ij}(a))$
   (b) $\varphi_{ij}(a) \in \mathcal{E}_k$
4. If $p \in \varepsilon^+_{ij}(a)$ and $s \in \ell_k(\varepsilon^+_{ij}(a))$, then $s \in \ell_{k+1}(p)$ and $p \in \mathcal{P}_{k+1}$

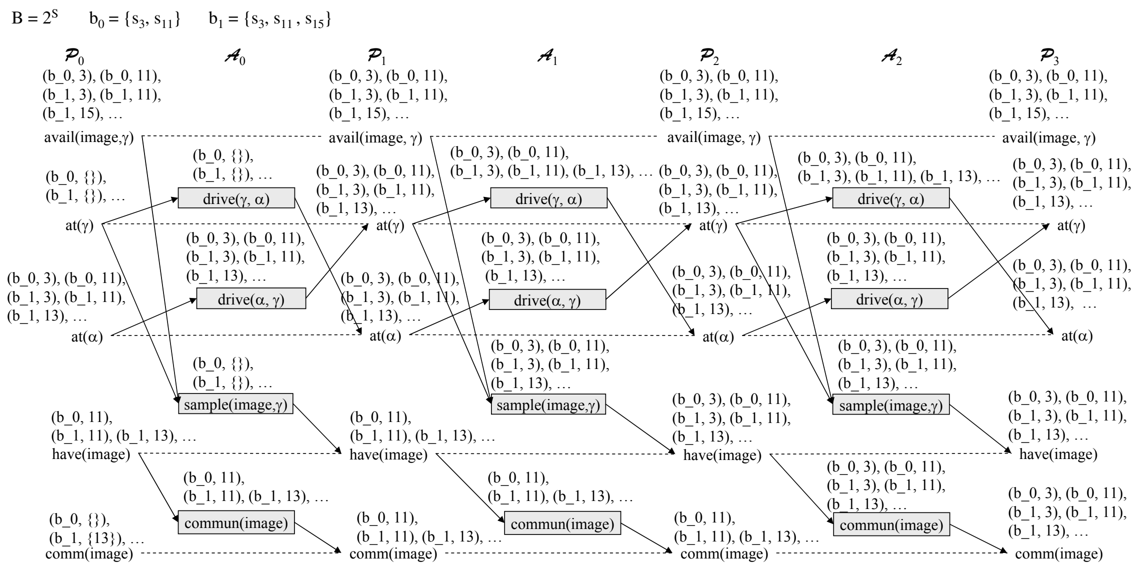
As stated, the *LUG* is a kind of SAG. The *LUG* is an efficient representation of a set of planning graphs built for classical planning with conditional effects. However, the *LUG* is more than that: the *LUG* is a full-fledged planning graph for belief-space planning. That is, I derive heuristics for belief-space from it by computing it for every search node. The level heuristic for the *LUG* is an estimate of the belief-space distance between the belief b and the goal G, $\text{argmin}_k\, b = \bigcap_{p \in G} \ell_k(p)$.

As I noted before, this heuristic is similar in structure to both the classical SAG and traditional planning graph level heuristic. The heuristic does not amortize graph construction effort across search nodes, but, it does use the SAG technique to efficiently build and reason about a set of planning graphs. That is, the set equivalence check reasons about a set of planning graphs (one for each $s \in b$) using only the single graph *LUG*(b). I introduced Figure 69 as an example of the SAG for classical planning. It is possible to re-interpret it as an example of the *LUG*. The graph, *LUG*(b), depicted in Figure 69 is built for the belief, b, that contains every state in the rover example.

I generalize the *LUG* to its state agnostic version by analogy to the state agnostic generalization of the classical planning graph. I introduce a new label syntax to track which sources reach vertices. In this case, sources are belief states. A further complication arises in that the *LUG* already defines its own labels. In order to complete the generalization, I represent labels as pairs of belief states and states. That is, the state agnostic *LUG* is a set of *LUG*s and each label denotes which states in which *LUG*s (distinguished by their source belief states) reach different graph vertices. Each pair in the set represented by a label signifies which *LUG* reaches the graph vertex, and which state of the *LUG* source reaches the vertex.

This state agnostic *LUG* is a labeled graph built for a scope $\mathcal{S}$ containing the set of belief states B. The following holds for every $b \in B$, and $s \in b$:

1. If p holds in s, then $(b, s) \in \ell_0(p)$ and $p \in \mathcal{P}_0$
2. If $(b, s) \in \ell_k(p)$ for every $p \in \rho_e(a)$, then:
    (a) $(b, s) \in \ell_k(a)$
    (b) $a \in \mathcal{A}_k$
3. If $a \in \mathcal{A}_k$ and $(b, s) \in \ell_k(p) \cap \ell_k(a)$ for every $p \in \rho_{ij}(a)$, then:
    (a) $(b, s) \in \ell_k(\varphi_{ij}(a))$
    (b) $\varphi_{ij}(a) \in \mathcal{E}_k$



B = 2^S b_0 = {s_3, s_11} b_1 = {s_3, s_11, s_15}
P_0
A_0
P_1
A_1
P_2
A_2
P_3
avail(image,γ)
at(γ)
at(α)
have(image)
comm(image)
drive(γ, α)
drive(α, γ)
sample(image,γ)
commun(image)
(b_0, 3), (b_0, 11), (b_1, 3), (b_1, 11), (b_1, 15), …
(b_0, {}), (b_1, {}), …
(b_0, 3), (b_0, 11), (b_1, 3), (b_1, 11), (b_1, 13), …
(b_0, 11), (b_1, 11), (b_1, 13), …
(b_0, {}), (b_1, {13}), …

Figure 71. Graph structure of a SAG, representing a set of *LUG*.

4. If $p \in \varepsilon^{+}_{ij}(a)$ and $(b, s) \in \ell_k(\varphi_{ij}(a))$, then $(b, s) \in \ell_{k+1}(p)$ and $p \in \mathcal{P}_{k+1}$

Intuitively, the label of a proposition, p, at level k represents a set of (b, s) pairs where *LUG*(b) considers the proposition k-reachable from state s in belief state b.

Figure 71 shows the state agnostic generalization of the *LUG* for the rovers problem. The labels contain pairs (b, s) for each belief state $b \subseteq S$ and $s \in b$, but the figure depicts the portion of the labels for two belief states: $b_0 = \{s_3, s_{11}\}$ and $b_1 = \{s_3, s_{11}, s_{15}\}$. Belief state b_0 is the initial belief state of the rover example, and b_1 results from applying commun$(image)$ to b_0. There are many labels that contain pairs that involve the same state for different belief states, such as ℓ_0(avail(image, gamma)), which contains $(b_0, 3)$ and $(b_1, 3)$.

It is possible to optimize this representation by eliminating the distinction between belief states in labels. Introducing these pairs of belief states and states is sufficient for representing arbitrary extensions of the planning graph to belief space. The *LUG*, however, does not require this mechanical scheme. Intuitively, the propagation rules of the *LUG* depend only upon properties of world states (as opposed to properties of belief states). $SLUG$ exploits this: $SLUG(B)$ represents a set (***LUG***$(b), b{\in}B$) for the price of a single element (***LUG***(b^*)). An, optimized, State Agnostic Labeled Uncertainty Graph, $SLUG(B)$, is a labeled graph built for a set of belief states B. The structure is equivalent to a particular ***LUG*** where $b^* = \bigcup_{b \in B} b$, such that $SLUG(B) =$***LUG***(b^*).

Figure 69 also depicts the $SLUG$ for every belief state in the rover example because $b^* = \bigcup_{b \in B} b = \bigcup_{b \in B} \bigcup_{s \in b} s = S$ (i.e., the union of all belief states is the set of states).

Any query concerning ***LUG***(b), for $b \in B$, can be answered using $SLUG(B)$ and the following rule for propositions (similarly for effects and actions): $p \in \mathcal{P}_k(b)$ iff: $p \in \mathcal{P}_k(B)$ and $b \subseteq \ell_k(p)$. By analogy with the level heuristic for the SAG, it is possible to compute a level heuristic for the $SLUG$. The level heuristic for belief-space planning, extracted from $SLUG(B)$ for belief state b is defined as $\mathrm{argmin}_k\, b \subseteq \bigcap_{p \in G} \ell_k(p)$. Like how the SAG is used for classical planning, heuristics use subset checks with $SLUG$ labels. The level heuristic for b_0 is three and for b_1, three also. The goal proposition is not labeled with all of the states in either b_0 or b_1 until proposition layer three.

4. Stochastic Planning

The SAG for non-deterministic and classical planning captures a single planning graph for each state because the actions are deterministic. In stochastic planning, the $\mathcal{M}$***cLUG*** deals with actions with uncertain outcomes, meaning there is potentially several planning graphs per state (each corresponds to a different set of joint outcomes). It is possible to use the $\mathcal{M}$***cLUG*** to compute a heuristic for a belief state by first sampling a set of states from the belief state, and for each state sampling from the set of potential planning graphs.

Developing a state agnostic generalization of the $\mathcal{M}$***cLUG*** requires a data structure that can compute the heuristic for all probabilistic belief states (of which there an infinite number). A complication is that the $\mathcal{M}$***cLUG*** generates new and potentially different planning graphs for each state sampled from each belief state. Since it is unknown how many times a state will be sampled during a search, it seems reasonable to sample a fixed pool of planning graphs for each state and re-sample from this pool. It is possible to ensure that the planning graphs for every set of state samples is constructed by generating N planning graphs for each state, where N is the maximum number of samples per belief state. This means that, using ideas from the $SLUG$ (where labels describe sets of states, instead of pairs of belief states and states), the state agnostic $\mathcal{M}c$*LUG* captures $O(N|S|)$ planning graphs.

Consider the number of random numbers, used in this scheme, to construct a state agnostic planning graph. Each of the $N|S|$ planning graphs represented by the state agnostic graph will have at most k levels, with $O(A)$ actions per level. If each action has an uncertain outcome, there are on the order of $O(N|S||A|^k)$ random numbers needed to construct the SAG. By sampling a unique outcome of every action in each planning graph, it is possible to generate the pool of planning graphs for heuristics. Under this scheme, each label is a set of pairs $(s, x^n(s))$, denoting the condition when a state s and a set of action outcomes, represented by $x^n(s)$, can make the labeled vertex reachable. The state agnostic version of the $\mathcal{M}c$*LUG* for all $s \in \mathcal{S}$ is defined by:

1. If $p \in s$, then $\forall_{n=0...N-1}(s, x^n(s)) \in \ell_0(p)$ and $p \in \mathcal{P}_0$
2. If $(s, x^n(s)) \in \ell_k(p)$ for every $p \in \rho_e(a)$, then:
   (a) $(s, x^n(s)) \in \ell_k(a)$
   (b) $a \in \mathcal{A}_k$
3. If $a \in \mathcal{A}_k$, $(s, x^n(s)) \in \ell_k(p) \cap \ell_k(a)$ for every $p \in \rho_{ij}(a)$, and $\Phi_i(a)$ is sampled by $x^n(s)$ then:
   (a) $(s, x^n(s)) \in \ell_k(\varphi_{ij}(a))$

   (b) $\varphi_{ij}(a) \in \mathcal{E}_k$

4. If $p \in \varepsilon^+_{ij}(a)$ and $(s, x^n(s)) \in \ell_k(\varphi_{ij}(a))$, then $(s, x^n(s)) \in \ell_{k+1}(p)$ and $p \in \mathcal{P}_{k+1}$

Since each label is a set of pairs $(s, x^n(s))$, the label representation is dependent both on the number of states and the number of samples per state. Using the notion of "common samples", it is possible to reduce the size of the label representation and the cost of computing the SAG. We consider a planning graph called the common sample SAG ($\mathcal{CSSAG}$) to share action outcome samples between states. In the $\mathcal{CSSAG}$, labels are sets of pairs (s, x^n), where x^n refers to a history of action outcome samples that is independent of the starting state. A label containing the pairs (s, x^n) and (s', x^n), signifies whether starting in state s or s' and fixing the same outcomes to uncertain actions in planning graph, the labeled vertex is reachable. This reduces the number of random numbers needed from $O(N|S||A|^k)$ to $O(N|A|^k)$ (the same number as the $\mathcal{M}c\mathcal{LUG}$). The $\mathcal{CSSAG}$ is defined by:

1. If p holds in s, then $\forall_{n=0...N-1}(s, x^n) \in \ell_0(p)$ and $p \in \mathcal{P}_0$
2. If $(s, x^n) \in \ell_k(p)$ for every $p \in \rho_e(a)$, then:
   (a) $(s, x^n) \in \ell_k(a)$
   (b) $a \in \mathcal{A}_k$
3. If $a \in \mathcal{A}_k$, $(s, x^n) \in \ell_k(p) \cap \ell_k(a)$ for every $p \in \rho_{ij}(a)$, and $\Phi_i(a)$ is sampled by x^n then:
   (a) $(s, x^n) \in \ell_k(\varphi_{ij}(a))$
   (b) $\varphi_{ij}(a) \in \mathcal{E}_k$
4. If $p \in \varepsilon^+_{ij}(a)$ and $(s, x^n) \in \ell_k(\varphi_{ij}(a))$, then $(s, x^n) \in \ell_{k+1}(p)$ and $p \in \mathcal{P}_{k+1}$

```
(define (domain rovers_stochastic)
  (:requirements :strips :typing)
  (:types location data)
  (:predicates
          (at ?x - location)
          (avail ?d - data ?x - location)
          (comm ?d - data)
          (have ?d - data))
  (:action drive
   :parameters (?x ?y - location)
   :precondition (at ?x)
   :effect (and (at ?y) (not (at ?x))))
  (:action commun
   :parameters (?d - data)
   :precondition (and)
   :effect (when (have ?d)
                 (probabilistic 0.8 (comm ?d))))
  (:action sample
   :parameters (?d - data ?x - location)
   :precondition (at ?x)
   :effect (when (avail ?d ?x)
                 (probabilistic 0.9 (have ?d))))
)
```

```
(define (problem rovers_stochastic1)
  (:domain rovers_stochastic)
  (:objects
     image - data
     alpha gamma - location)
  (:init (at alpha)
        (avail image gamma)
        (probabilistic 0.9 (have image)))
  (:goal (comm soil) 0.99)
)
```

Figure 72. PDDL description of stochastic planning formulation of rover problem.

Consider the extension of the rover example listed in Figure 72. There are probabilistic effects for the `sample(image, gamma)` and `commun(image)` actions, and a probabilistic initial belief state, denoting that `have(image)` is true with 0.9 probability. The corresponding $\mathcal{CSSAG}$ for this problem is shown in Figure 73. The labels are depicted in a short hand that describes a set of pairs by two sets, the states, and the samples, where every pair of elements from the two sets is in the label. For example, the pairs $(s_2, x^0), (s_4, x^0), ..., (s_2, x^1), ...$ are in the label ℓ_0(`at(gamma)`). Similarly, the pairs $(s_9, x^0), ..., (s_8, x^2)$ are in the label ℓ_1(`have(image)`). The labeled effects, show which samples sample the outcome where the effect occurs, such as $\{x^0, x^1, x^2\}$ for the effect φ_{00}(`sample(image, gamma)`).

Using the $\mathcal{CSSAG}$ to evaluate a heuristic for a belief state b involves sampling a set of N states from the belief state. Each sampled state s is paired with a sample identifier x^n, such that a set

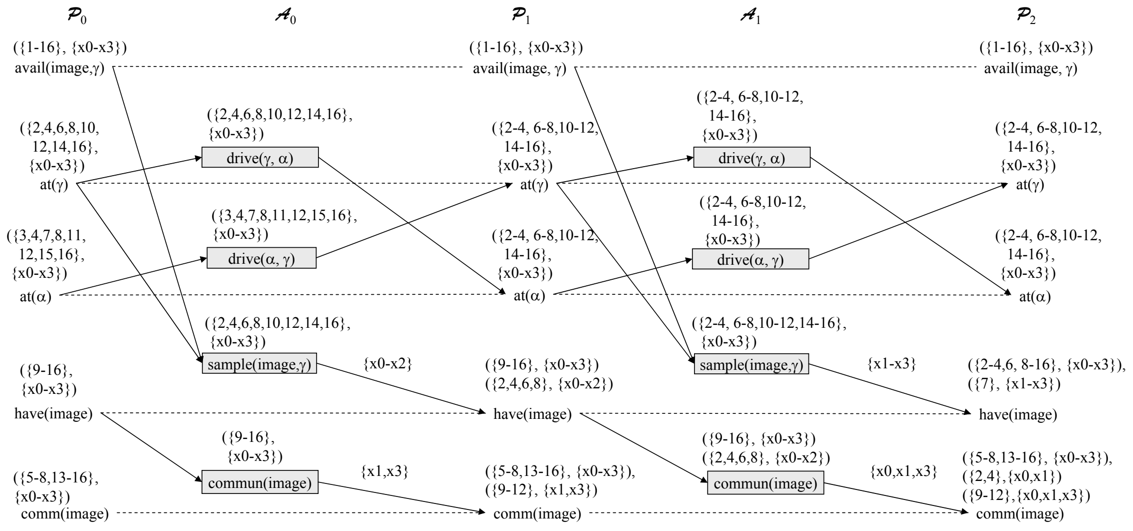


Figure 73. Graph structure of a $\mathcal{CSSAG}$.

$\{(s, x^n)\}_{n=0}^{N-1}$ identifies the $\mathcal{M}cLUG(b)$. For instance, the level heuristic is the first level k where the proportion of state and sample pairs (sampled from the belief state) that reach the goal exceeds the goal satisfaction threshold, $\frac{|\bigcap_{p \in G} \ell_k(p) \cap \{(s,x^n)\}_{n=0}^{N-1}|}{N} \geq \tau$. For example, the level heuristic for the initial belief state $\{0.1 : s_3, 0.9 : s_{11}\}$ to reach the goal of the stochastic rover formulation is computed as follows. If $N = 4$, we may sample states from the initial belief state to obtain $\{(s_3, x^0), (s_{11}, x^1), (s_{11}, x^2), (s_{11}, x^3)\}$. The intersection between the label of the goal proposition comm(image) and the state samples is $\{\}$ at level zero, $\{(s_{11}, x^1), (s_{11}, x^3)\}$ at level one, and $\{(s_{11}, x^1), (s_{11}, x^3)\}$ at level two. Thus, the probability of reaching the goal is 0/4 = 0 at level zero, 2/4 = 0.5 at level one, and 2/4 = 0.5 at level two. Without expanding the $\mathcal{CSSAG}$ further, it is possible to say the level heuristic is at least 2.

5. SAG Relaxed Plans

There two fundamental techniques that I identify for extracting relaxed plans from state agnostic planning graphs to guide $POND$'s AO* search. The first, called the SAG relaxed plan, extracts relaxed plans from the state agnostic graph during search, upon generating each search node. The second, called a global relaxed plan, extracts the relaxed plan for each state (or state and sample pair) before search, and during search combines the necessary relaxed plans to evaluate specific search nodes.

SAG Relaxed Plans: Recall that I extract a belief-space relaxed plan from *LUG*(b) ($\mathcal{M}c$*LUG*(b)) by requiring that, for any $s \in b$ ($(s, x^n) \sim b$), the set of vertices whose labels' evaluate to true for s ((s, x^n))represents a classical relaxed plan. Previous chapters demonstrate the procedure for efficiently extracting such a plan. The relaxed plan can be viewed as ignoring negative interactions between states in a belief state in addition to ignoring negative interactions between actions. I extend this to extracting belief-space relaxed plans from $SLUG(\mathcal{S})$, for any $b \subseteq \mathcal{S}$, by intersecting the labels of the $SLUG$ with b before using them in the above extraction procedure. As with classical relaxed plans, the heuristic value of a belief-space relaxed plan is the number of action vertices it contains. The same holds for the $\mathcal{C}SSAG(\mathcal{S})$, where $\mathcal{S} = \{(s, x^n) | s \in S, 0 \leq n \leq N-1\}$ by taking the intersection of a set of state-sample pairs with labels.

Global Relaxed Plans: Much like how the $SLUG$ can be used to represent a set of *LUG*s with labels indicating which states are relevant to which planning graph vertices, it is possible to do the same for relaxed plans. The idea is to extract a relaxed plan to support the goals from the belief state $b = \mathcal{S}$ that captures the entire scope of the $SLUG$ (or $\mathcal{C}SSAG$). Each action in this global relaxed plan is labeled by the subset of its label $\ell_k^{RP(\mathcal{S})}(a) \subseteq \ell_k(a)$ that denotes the states that use it to support the goal. Then, to evaluate the heuristic for a specific belief state $b' \subseteq \mathcal{S}$, I count the number of actions in the global relaxed plan that are used for the states in b' (i.e., those actions where $b' \cap \ell_k^{RP(\mathcal{S})}(a) \neq \emptyset$).

The trade-off between the SAG relaxed plan and the global relaxed plan, is very similar to the trade-off between using a SAG and a standard planning graph: *a priori* construction cost must be amortized over search node expansions. However, there is another subtle difference between the relaxed plans computed between the two techniques. Recall that relaxed plan extraction is guided by a simple heuristic (in the absence of cost information) that prefers to support propositions with supporters that contribute more possible worlds. In the SAG relaxed plan, this set of possible worlds contains exactly those possible worlds that are needed to evaluate a search node. In the global relaxed plan, the set of possible worlds (e.g., any subset of S) used to choose supporters may be much larger than the set used to evaluate any given search node. By delaying the relaxed plan extraction to curtail it to each search node, the SAG relaxed plan can be more informed. By committing to a global relaxed plan before search (without knowledge of the actual search nodes), the heuristic may make poor choices in relaxed plan extraction and be less informed. Despite the possibility of poor relaxed plans, the global relaxed plan is quite efficient to compute for every search node (modulo the higher extraction cost).

6. Choosing the SAG Scope

The choice of scope has a great impact on the performance of any approach based on SAG. Using fewer states in the scope almost always requires less computation per graph. While not every label function becomes smaller, the aggregate size almost always decreases. Restricting the scope, however, prevents the SAG from representing the planning graphs for states so excluded. If such states are visited in search, then a new SAG will need to be generated to cover them. All of the approaches based on SAG can be seen as a particular strategy for covering the set of all states with shared graphs. I define 4 isolated points in that spectrum: Global-SAG, Reachable-SAG, Child-SAG, and the PG (Node-SAG).

Global-SAG: A Global-SAG is a single graph for an entire planning episode. The scope is taken to be the set of all states (or state and sample pairs); that is, Global-SAG uses the degenerate partition containing only the set of all states (or state and sample pairs).

Reachable-SAG: A Reachable-SAG is also a single graph for an entire planning episode. A normal planning graph (*LUG*) is constructed from the initial belief state of the problem; all states consistent with the last level form the set from which the Reachable-SAG is built. That is, states are partitioned into two groups: definitely unreachable, and possibly reachable. A graph is generated only for the latter set of states.

Child-SAG: A Child-SAG is a graph built for the set of children of the current node. This results in one graph per non-leaf search node. That is, Child-SAG partitions states into sets of siblings. While this is still an exponential number of graphs, the reduction in computation time, relative to Node-SAG, is still significant.

Node-SAG: A Node-SAG is built for the smallest conceivable scope: the current search node. In some situations, label propagation is useless, and therefore skipped. That is, Node-SAG is just a name for the naive approach of building a planning graph for every search node.

7. Empirical Evaluation

The empirical evaluation is divided into two subsections for an internal and external evaluation. The internal evaluation covers i) the effect of using different scopes, and ii) the effect of using SAG or global relaxed plans. The external evaluation compares several of the better techniques (from the internal evaluation) to several state of the art planners, including results from Fifth International Planning Competition (IPC).

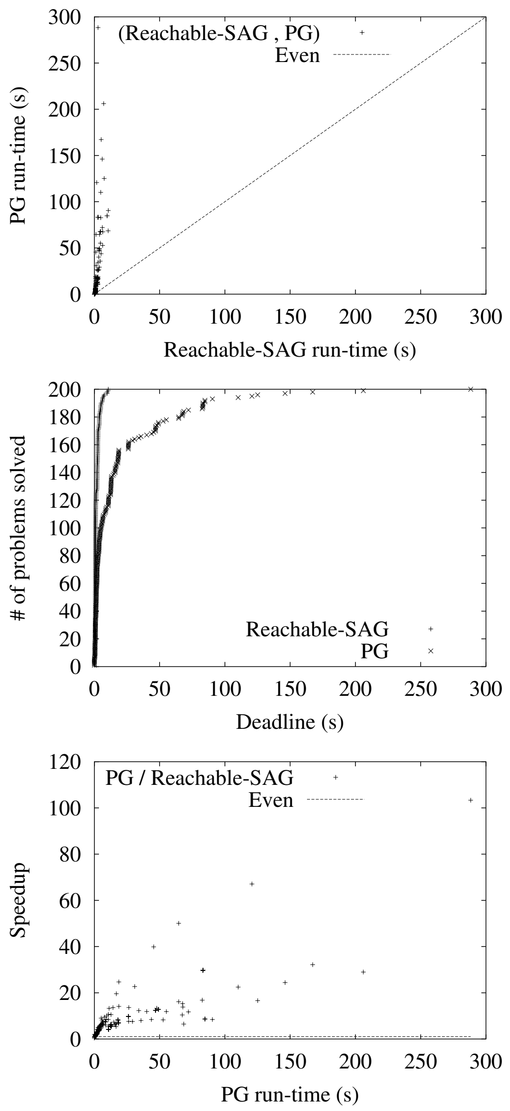
In order to qualitatively evaluate the techniques, I developed them within POND. I implemented all of the SAG strategies (Node, Child, Reachable, Global) within POND. The degenerate case, Node-SAG, simply uses *LUG* or $\mathcal{M}cLUG$. The other three cases are implemented with respect to the (optimized) state agnostic version of *LUG*: $SLUG$, or $\mathcal{C}SSAG$. The implementation uses the relaxed plan based heuristic, identical to the relaxed plan heuristic developed in Chapter 4 for the *LUG*, or in Chapter 6 for the $\mathcal{M}c$*LUG*.

I ran tests across a wide variety of benchmark problems in belief-space planning. All problems in the POND distribution (from Chapter 3) were attempted; I further augmented the test set with all benchmarks from the 2000, 2002, and 2004 planning competitions. The latter problems are classical planning problems, which we include in order to gauge the possible benefits of applying SAG to classical planners.

7.1. Internal Comparison. Within the internal comparison, I discuss results for different choices of scope and methods of computing the relaxed plan heuristic. The results were collected in four different groups: IPC classical planning problems, POND's non-deterministic problems, IPC non-deterministic conformant planning problems, and POND's conformant probabilistic problems (from Chapter 6). The choices for SAG scope are evaluated on every group, and the choices for relaxed plan are evaluated on the last two groups.

The choices for SAG scope are evaluated on four test sets, described below. All but the first two sets were evaluated on different machines with different characteristics. The first two test sets are evaluated under the same methodology.

IPC Classical Planning & POND Non-Deterministic Problems: There are several hundred problems in the IPC test sets, so I imposed relatively tight limits on the execution (5 minutes on a P4 at 3.06 GHz with 900 MB of RAM) of any single problem. I exclude failures due to these limits from the figures. In addition, I sparsely sampled these failures with relaxed limits to ensure that my conclusions were not overly sensitive to the choice of limits. Up until the point where physical memory is exhausted, the trends remain the same.



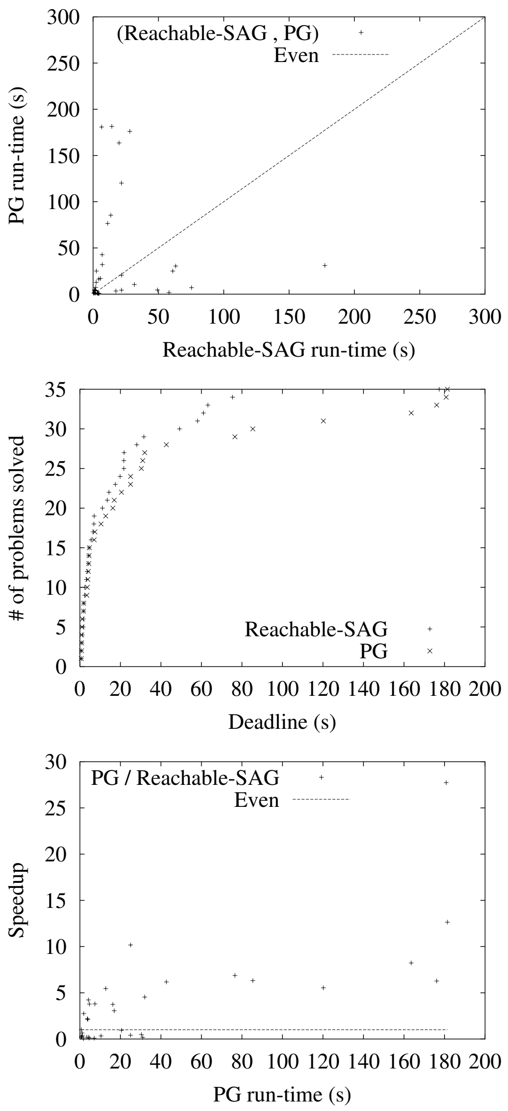


Figure 74. Reachable-SAG (using $SLUG$) vs. PG (using LUG), Non-Deterministic Problems

Figure 75. Reachable-SAG (using $SLUG$) vs. PG (using LUG), Classical Problems

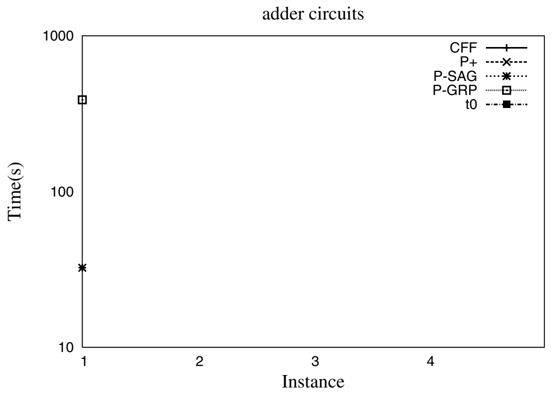
The measurement for each problem is the total run time of the planner, from invocation to exit. I have only modified the manner in which the heuristic is computed; despite this, I report total time to motivate the importance of optimizing heuristic computation. It should be clear, given that I achieve large factors of improvement, that time spent calculating heuristics is dominating time spent searching.

Figures 74 and 75 provide several perspectives on the merits of Reachable-SAG in non-deterministic and classical planning. The figure omits Global-SAG and Child-SAG for clarity. The former, Global-SAG, is dominated; the mode wastes significant time projecting reachability for unreachable states. The latter, Child-SAG, improves upon the PG approach in virtually all problems. However, that margin is relatively small, so I prefer to depict the current standard in the literature, the PG approach. The top graph in each figure is a scatter-plot of the total running times. The line "y=x" is plotted, which plots identical performance. The middle graph in each figure plots the number of problems that each approach has solved by a given deadline. The bottom graph in each figure offers one final perspective, plotting the ratio of the total running times.

The scatter-plots reveal that Reachable-SAG always outperforms the PG approach. Moreover, the boost in performance is well-removed from the break-even point. The deadline graphs are similar in purpose to plotting time as a function of complexity: rotating the axes reveals the telltale exponential trend. However, it is difficult to measure complexity across domains. This method corrects for that at the cost of losing the ability to compare performance on the same problem. We observe that, with respect to any deadline, Reachable-SAG solves a much greater number of planning problems. Most importantly, Reachable-SAG out-scales the PG approach. When we examine the speedup graphs, we see that the savings grow larger as the problems become more difficult.

POND treats classical problems as problems in belief-space; naturally, this is very inefficient. Despite this, Reachable-SAG still produces an improvement on average. While the scatter-plots reveal that performance can degrade, it is still the case that average time is improved: mostly due to the fact that as problems become more difficult, the savings become larger.

In light of this, my colleagues have made a preliminary investigation in comparing SAG to state of the art implementations of classical planning graphs. In particular, Hoffmann and Nebel (2001) go to great lengths to build classical planning graphs as quickly as possible, and subsequently extract a relaxed plan. They attempted to compete against that implementation with a straightfor-



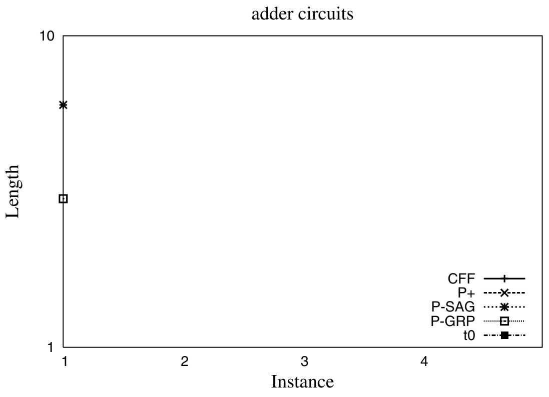
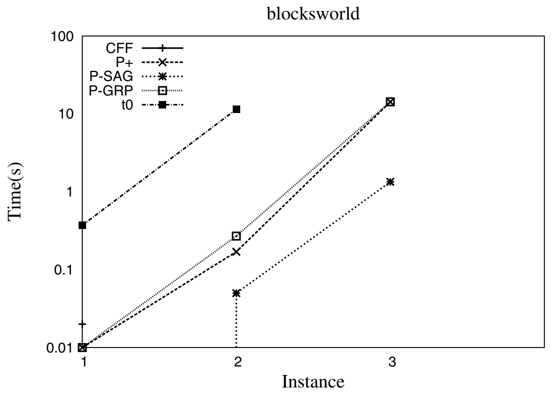


Figure 76. Run times (s) and Plan lengths IPC5 Adder Instances.



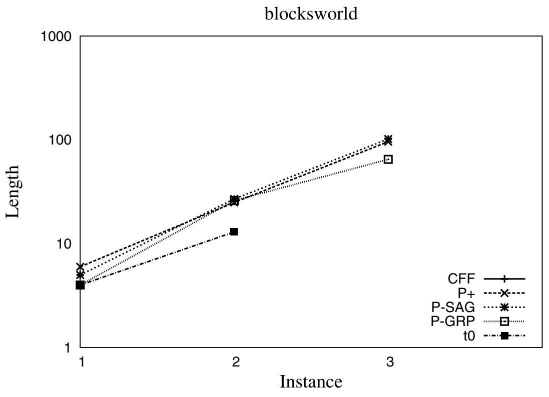
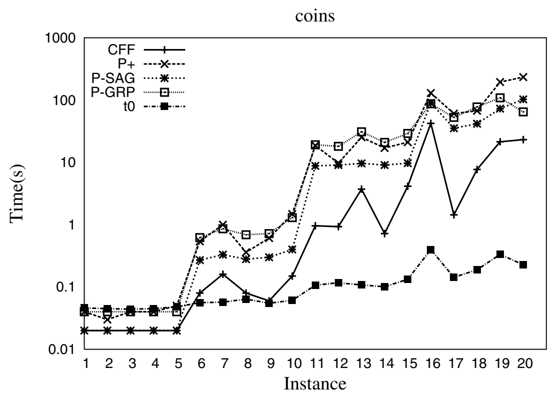


Figure 77. Run times (s) and Plan lengths IPC5 Blocksworld Instances.

ward implementation of SAG. They ran trials of greedy best-first search using SAG relaxed plan heuristic against using same heuristic with the same heuristic as the Reachable-SAG strategy. Total performance was improved, sometimes doubled, for the Rovers domain; however, in most other benchmark problems, the relative closeness of the goal and the poor estimate of reachability prohibited any improvement. Of course, per-node heuristic extraction time (i.e. ignoring the time it takes to build the shared graph) was always improved, which motivates an investigation into more sophisticated graph-building strategies than Reachable-SAG.

IPC5 Non-Deterministic Conformant Planning Domains: Figures 76, 77, 78, 79, 80, and 81



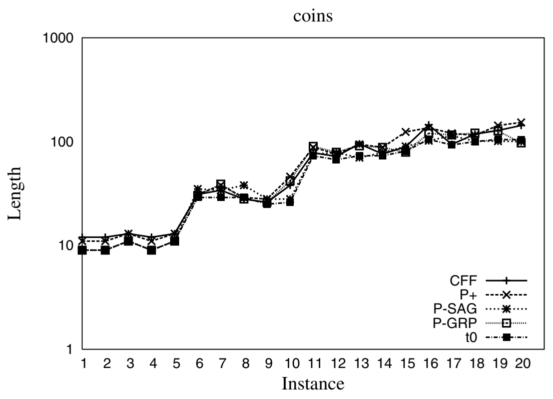
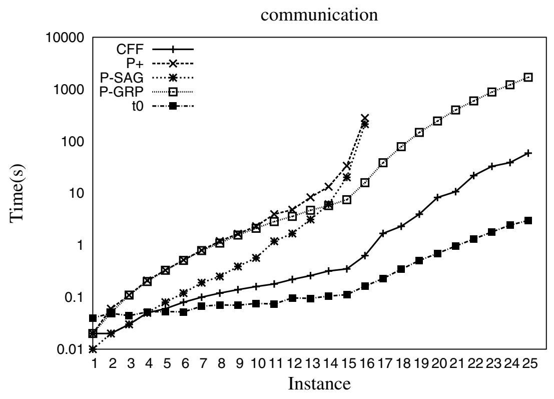


Figure 78. Run times (s) and Plan lengths IPC5 Coins Instances.



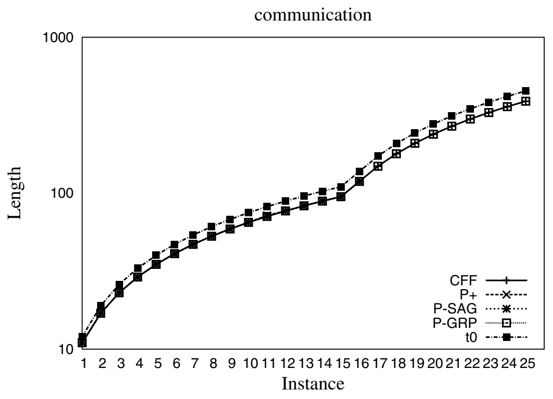
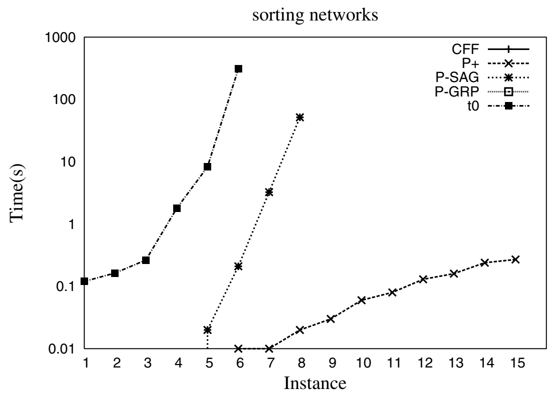


Figure 79. Run times (s) and Plan lengths IPC5 Comm Instances.



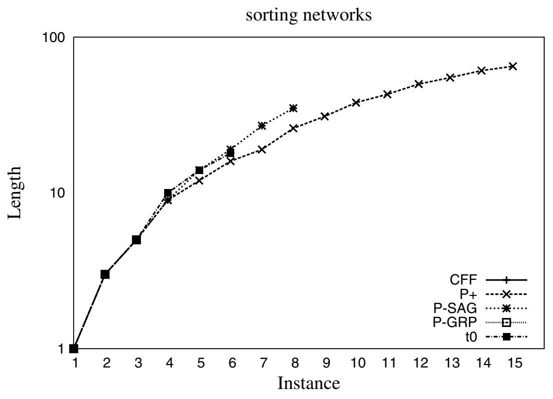


Figure 80. Run times (s) and Plan lengths IPC5 Sortnet Instances.

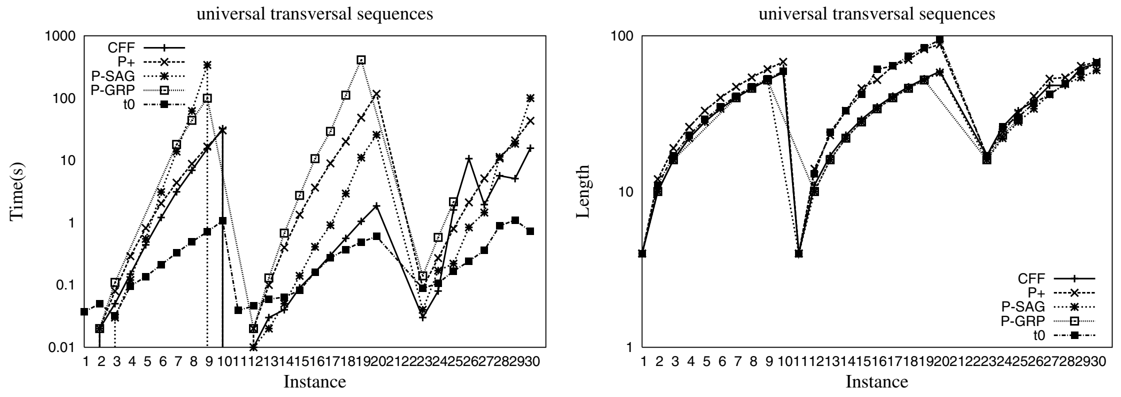


Figure 81. Run times (s) and Plan lengths IPC5 UTS Instances.

show total time in seconds and plan lengths for six domains used for the Fifth International Planning Competition non-deterministic planning track. The first domain, called adder circuits, involves actions describing logic gates that must be sequenced to synthesize an n-bit adder. The second domain, called blocksworld, is similar to the classical planning domain, but includes initial state uncertainty where any reachable state is part of the initial belief state. The third domain, called coins, involves an agent in a unknown location of a building with multiple floors needing to collect all the coins in the building. The fourth domain, called communication, involves synthesizing a network access protocol for a noisy network. The fifth domain, called sorting networks, involves sequencing a set of gates to sort an a priori unknown array of binary numbers. The sixth domain, called universal traversal sequences, is a network problem where a message must be broadcast from an unknown node to all others.

In the next section, I will discuss comparisons with other planners. In this section, I concentrate on the three versions of $POND$ that were entered in the competition. The three versions of $POND$, denoted as P-+, P-SAG, and P-GRP, use an enforced hill-climbing search algorithm, instead of AO*, and differ with respect to the heuristics used. P-SAG uses the SAG relaxed plan, as did the previously discussed sets of results. P-GRP uses the global relaxed plan. P-+ uses both heuristics, plus an additional cardinality heuristic (measuring belief state size).

The way that P-+ uses the three heuristics to guide enforced hill-climbing requires a brief introduction to enforced hill-climbing search. Enforced hill-climbing is a local search that attempts to extend a plan prefix. Each extension is evaluated with a heuristic that measures the distance to the goal, and if an action decreases the distance it is chosen to extend the plan, and the next extension is considered. If no extension decreases distance to the goal, a breadth-first search is used to find an extension (that is possibly multiple steps) that decreases the heuristic estimate of the distance to the goal. P-+ uses the cardinality heuristic first, and if it cannot find an extension that decreases distance to the goal, then it re-evaluates the nodes with the global relaxed plan. If the global relaxed plan does not identify any extension that decreases distance to the goal, then the SAG relaxed plan is used to re-evaluate the nodes. If no heuristic can find a better extension, then breadth-first search is used. In this manner, P-+ tries increasingly accurate, but costly, heuristics to re-evaluate the same plan extensions. The P-SAG and P-GRP use the respective SAG relaxed plan, or global relaxed plan, to evaluate plan extensions once before breadth-first search.

Looking at the results, there are several observations about the behavior of the different heuristics. P-SAG and P-GRP can solve the first adder problem, but P-+ cannot. P-SAG is the fastest technique for solving blocksworld and coins problems, but P-GRP is fastest for communication problems. P-+ is fastest for sortnet problems (because it only uses cardinality), and P-SAG is the faster of the techniques actually using a relaxed plan heuristic. In sortnet, it is very costly to compute a global relaxed plan, making it run out of memory. In universal traversal sequences, it is not clear whether P-SAG or P-+ is better, but P-GRP is dominated. From these results, it appears that the SAG relaxed plan is better (consistently faster) than the global relaxed plan heuristic (with the exception of the communication domain). The reason that the global relaxed plan heuristic performs worse is that it can take quite a long time to compute it, and it can give inferior search guidance. In the cases where the global relaxed plan can be computed in reasonable time and provide enough guidance, it does better.

POND Probabilistic Conformant Planning Domains: Figures 82, 83, 84, 85, 86, 88, and 87 show total time in seconds and plan length results for the probabilistic conformant planning problems from Chapter 6. The figures compare the SAG relaxed plan (denoted SAG), the global relaxed plan (denoted GRP), the $\mathcal{M}c$*LUG*(denoted by the number of particles), and CPplan. I use 16 or 64 samples, as indicated in the legend of the figures for each domain, in each $\mathcal{C}SS$*AG* (used for SAG and GRP) or $\mathcal{M}c$*LUG* because these numbers proved best in the evaluation from Chapter 6. This version of $POND$ uses AO* search instead of enforced hill climbing.

Figure 82 shows that the $\mathcal{C}SS$*AG* and the SAG relaxed plan improves, if only slightly, upon the $\mathcal{M}c$*LUG* in Logistics p2-2-2. However, the $\mathcal{C}SS$*AG* with the global relaxed plan takes longer and finds much worse plans. Figure 83 shows similar results for Logistics p4-2-2, with the SAG relaxed plan performing considerably better – solving the problem where $\tau = 0.95$. Figure 84 shows results for Logistics p2-2-4 that demonstrate the improvements of using the SAG relaxed plan, finding plans for $\tau = 0.95$, where the $\mathcal{M}c$*LUG* could only solve instances where $\tau \leq 0.25$. In general, the global relaxed plan performs worse than the SAG relaxed plan both in terms of time and quality. However, the global relaxed plan scales better than the $\mathcal{M}c$*LUG* in some cases, despite poor quality solutions. Overall, using the $\mathcal{C}SS$*AG* is better than the $\mathcal{M}c$*LUG*.

Figure 85 shows results for the Grid-0.8 domain that indicate the $\mathcal{C}SS$*AG* greatly improves total planning time with both the SAG and global relaxed plans over the $\mathcal{M}c$*LUG*. However, Figure 86 shows the results are different for Grid-0.5. The $\mathcal{C}SS$*AG* heuristics perform much worse than the $\mathcal{M}c$*LUG*. A potential explanation is the way in which the $\mathcal{C}SS$*AG* chooses a pool of common action outcome samples to use in the planning graphs. The $\mathcal{M}c$*LUG* is more robust to the sampling because it re-samples the action outcomes for each belief state, where the $\mathcal{C}SS$*AG* re-samples the

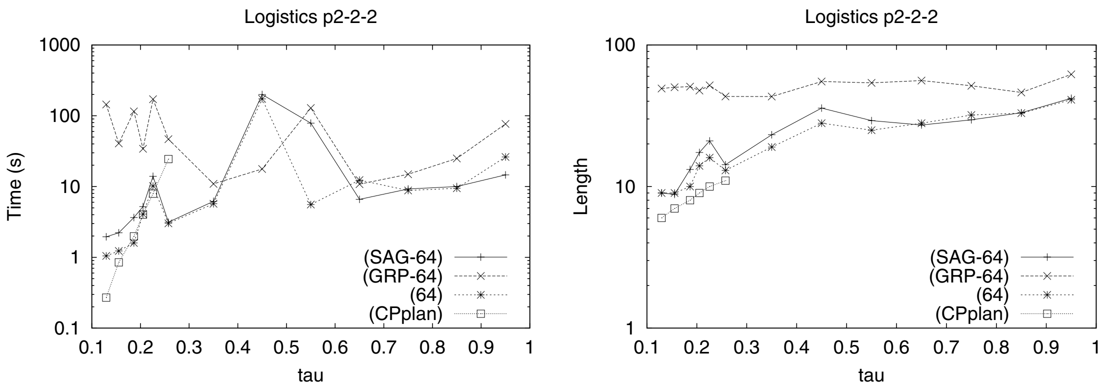


Figure 82. Run times (s) and Plan lengths vs. τ (log scale) for Logistics p2-2-2

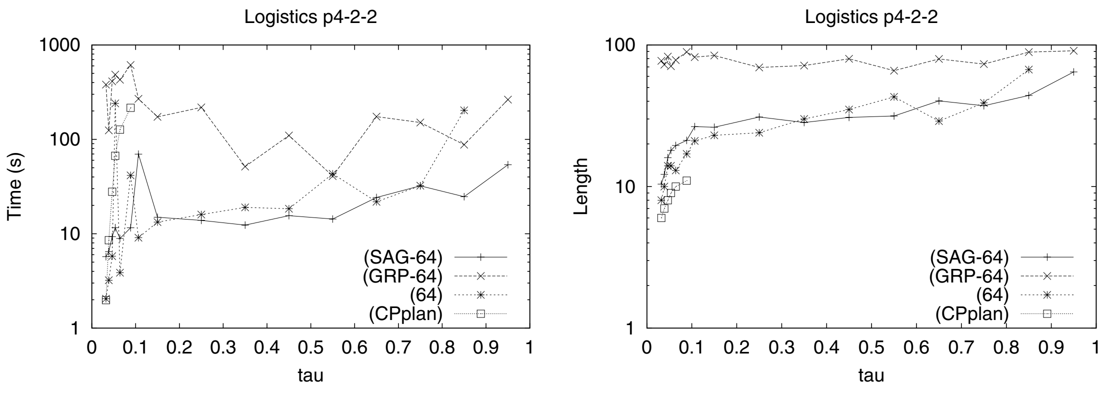


Figure 83. Run times (s) and Plan lengths vs. τ (log scale) for Logistics p4-2-2

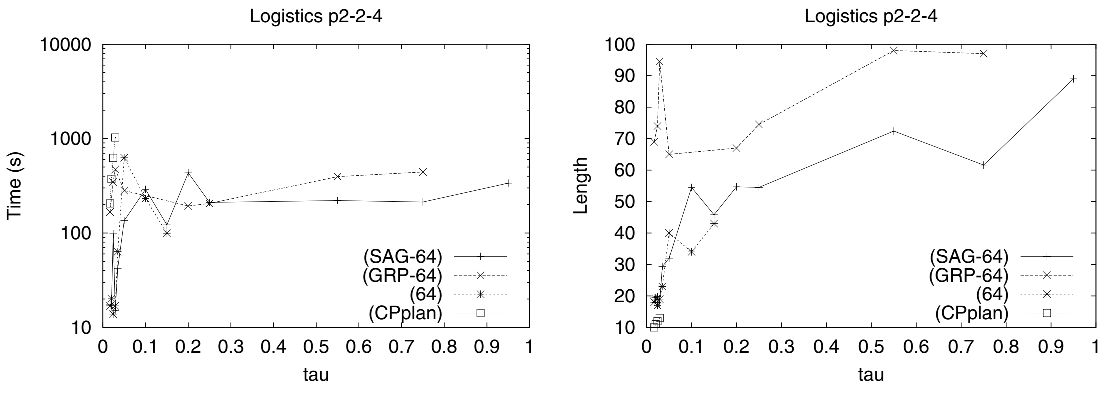


Figure 84. Run times (s) and Plan lengths vs. τ (log scale) for Logistics p2-2-4

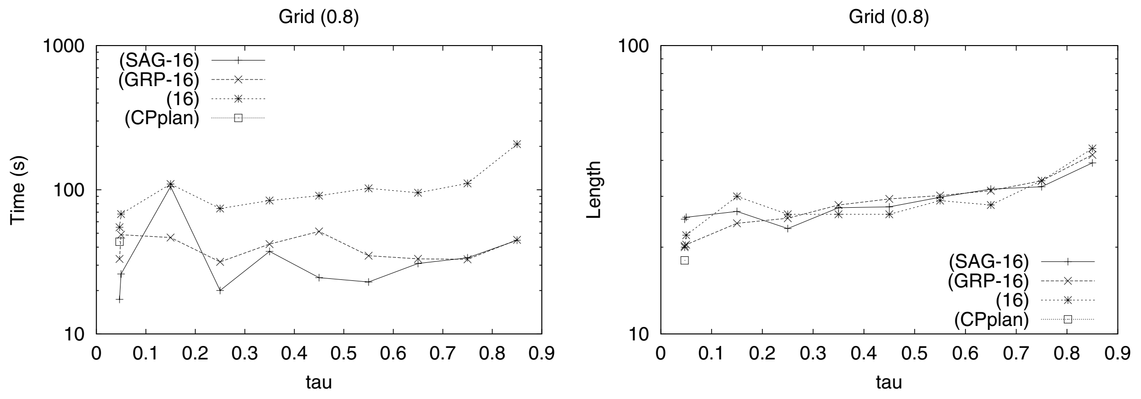


Figure 85. Run times (s) and Plan lengths vs. τ for Grid-0.8

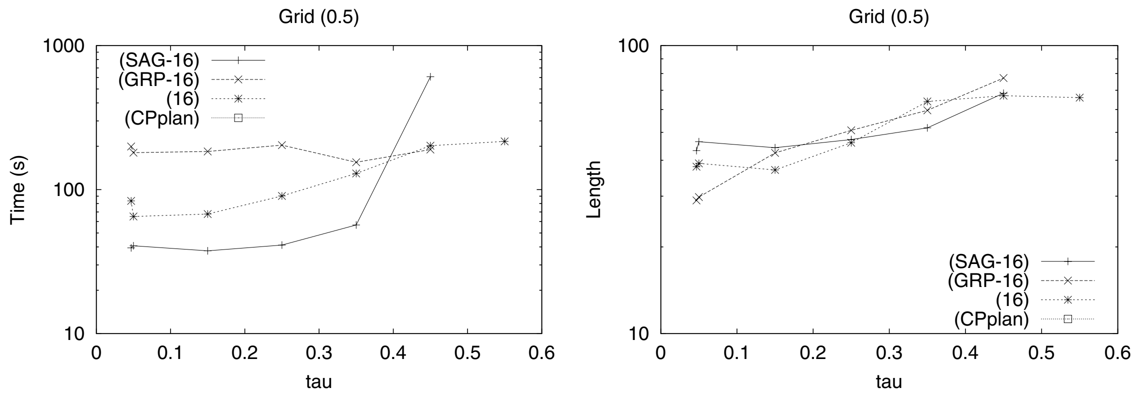


Figure 86. Run times (s) and Plan lengths vs. τ for Grid-0.5

action outcomes from the pre-sampled pool.

Figures 88 and 87 show results for the respective SandCastle-67 and Slippery Gripper domains, where the $\mathcal{M}cLUG$ outperforms the $\mathcal{C}SSAG$ with either relaxed plan heuristic. Similar to the Grid-0.5 domain, $\mathcal{C}SSAG$ has an impoverished pool of action outcome samples that does not plague the $\mathcal{M}cLUG$. Since these problems are relatively small, the cost of computing the $\mathcal{M}cLUG$ pales in comparison to the quality of the heuristic it provides.

Overall, the $\mathcal{C}SSAG$ is useful when it is too costly to compute a $\mathcal{M}cLUG$ for every search node, but it seems to provide less informed heuristics. Using the global relaxed plan generally takes

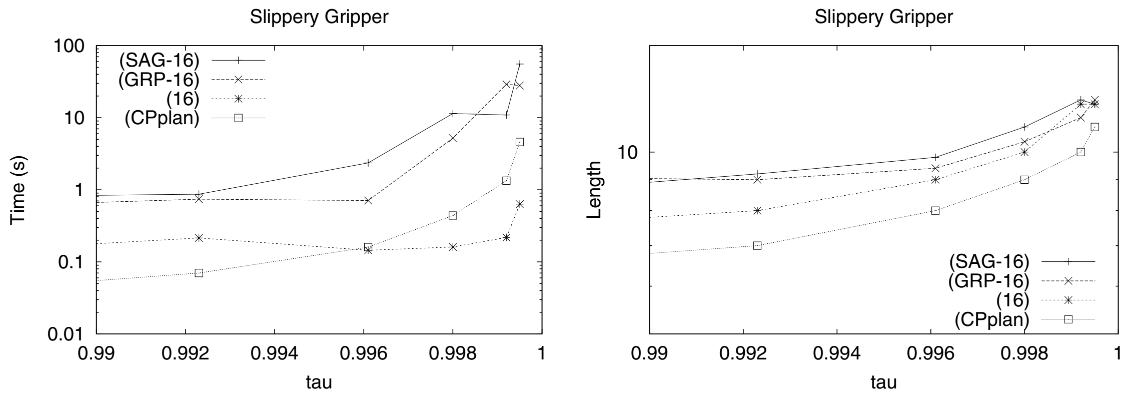


Figure 87. Run times (s) and Plan lengths vs. τ for Slippery Gripper.

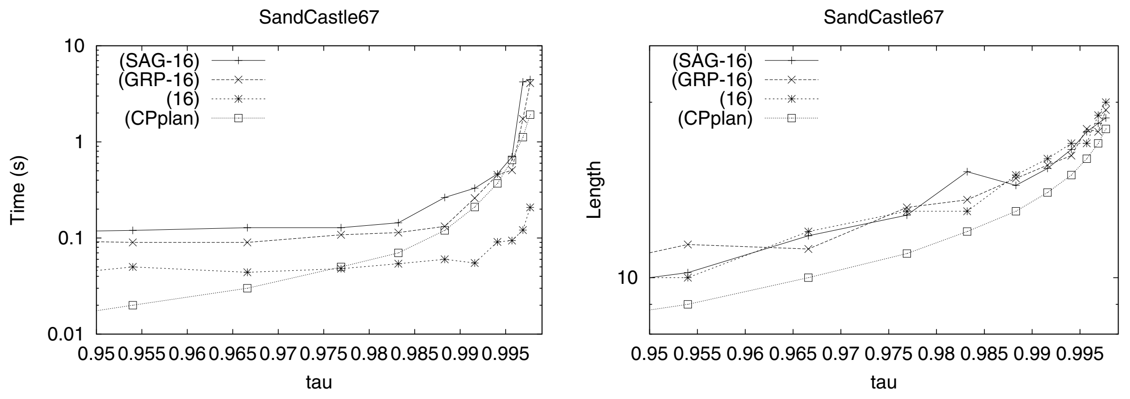
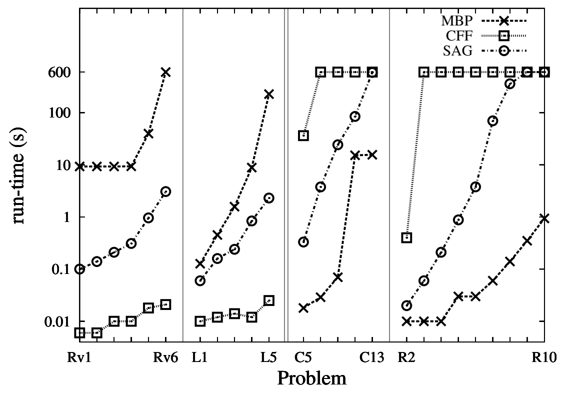


Figure 88. Run times (s) and Plan lengths vs. τ for SandCastle-67.

as much time to find a plan as does the SAG relaxed plan, but finds much worse plans.

7.2. External Comparison. Like the internal evaluation, this section uses several groups of test problems to evaluate the SAG: POND's non-deterministic planning problems (from Chapter 3) and the Fifth International Planning competition non-deterministic track domains. The results on POND's conformant probabilistic planning problems are similar to the results described in Chapter 6, and compared to CPplan performance only improves.



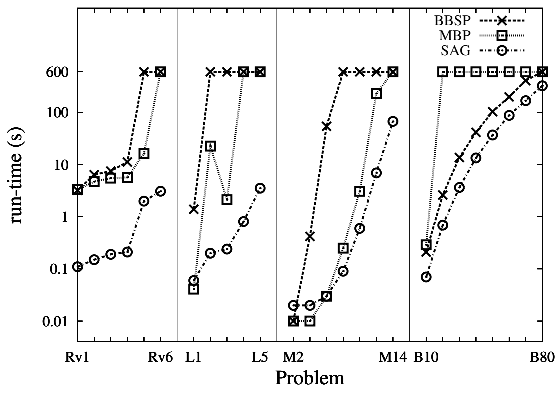


Figure 89. Comparison of planners on conformant (left) and conditional (right) domains. Four domains appear in each plot. The conformant domains include Rovers (Rv1-Rv6), Logistics (L1-L5), Cube Center (C5-C13), and Ring (R2-R10). The conditional domains are Rovers (Rv1-Rv6), Logistics (L1-L5), Medical (M2-M14), and BTCS (B10-B80).

POND's Non-Deterministic Domains: I made an external comparison of POND with several of the best conformant: KACMBP (Bertoli and Cimatti, 2002) and CFF (Brafman and Hoffmann, 2004), and conditional planners: MBP (Bertoli *et al.*, 2001) and BBSP (Rintanen, 2005). Based on the results of the internal analysis, I used belief-space relaxed plans extracted from a common $SLUG$, using the Reachable-SAG strategy. I denote this mode of POND as "SLUG" in Figure 89. The tests depicted in Figure 89 were allowed 10 minutes on a P4 at 2.8 GHz with 1GB of memory. The planners we used for these comparisons require descriptions in differing languages. I ensured that each encoding had an identical state space; this required us to use only boolean propositions in our encodings.

I used the conformant Rovers and Logistics domains as well as the Cube Center and Ring domains (Bertoli and Cimatti, 2002) for the conformant planner comparison in Figure 89. These domains exhibit two distinct dimensions of difficulty. The primary difficulty in Rovers and Logistics problems centers around *causing* the goal. The Cube Center and Ring domains, on the other hand, revolve around *knowing* the goal. The distinction is made clearer if we consider the presence of an oracle. The former pair, given complete information, remains difficult. The latter pair, given complete information, becomes trivial, relatively speaking.

I see my heuristic as a middle-ground between KACMBP's cardinality based heuristic and CFF's approximate relaxed plan heuristic. In the Logistics and Rovers domains, CFF dominates, while KACMBP becomes lost. The situation reverses in Cube Center and Ring: KACMBP easily discovers solutions, while CFF wanders. Meanwhile, by avoiding approximation and eschewing cardinality in favor of reachability, POND achieves middle-ground performance on all of the problems.

I devised conditional versions of Logistics and Rovers domains by introducing sensory actions. I also drew conditional domains from the literature: BTCS (Weld *et al.*, 1998) and a variant of Medical (Petrick and Bacchus, 2002). Our variant of Medical splits the multi-valued stain type sensor into several boolean sensors.

The results (Figure 89) show that POND dominates the other conditional planners. This is not surprising: MBP's and BBSP's heuristic is belief state cardinality. Meanwhile, POND employs a strong, yet cheap, estimate of reachability (relaxed plans extracted from $SLUG$, in Reachable-SAG mode). MBP employs greedy depth-first search, so the quality of plans returned can be drastically poor. The best example of this in our results is instance Rv4 of Rovers, where the max length branch of MBP requires 146 actions compared to 10 actions for POND.

International Planning Competition: In the conformant planning track of the Fifth IPC, POND was entered with three variations, previously described. POND was the only planner to solve instances in the adder domain, and outperformed all other planners in the blocksworld and sortnet domains. POND was competitive, but slower in the coins, communication, and universal traversal sequences domains. Overall, POND exhibited good performance across all domains, as a domain-independent planner should.

8. Related Work

I have already noted that this work is a generalization of the *LUG*, which efficiently exploits the overlap in the planning graphs of members of a belief state. Liu *et al.* (2002) have explored issues in speeding up heuristic calculation in HSP. Their approach utilizes the prior planning graph to improve the performance of building the current planning graph (the rules which express the dynamic program of HSP's heuristic correspond to the structure of a planning graph). I set out to perform work ahead of time in order to save computation later; their approach demonstrates how to boost performance by skipping re-initialization. Also in that vein, Long and Fox (1999) demonstrate techniques for representing a planning graph that take full advantage of the properties of the planning graph. I seek to exploit the overlap between different graphs, not different levels. Liu *et al.* (2002) seek to exploit the overlap between different graphs as well, but limit the scope to graphs adjacent in time. Another way of avoiding the inefficiencies in repeated construction of planning graphs is to do the reachability computation in the backward direction (Kambhampati *et al.*, 1997). However, I note that state of the art progression planners typically do reachability analysis in the forward direction. Work on greedy regression graphs (McDermott, 1999) as well as the GRT system (Refanidis and Vlahavas, 2001), can be understood this way.

9. Conclusion

A common task in many planners is to compute a set of planning graphs. The naive approach fails to take account of the redundant sub-structure of planning graphs. I developed the state agnostic graph (SAG) as an extension of prior work on the labeled uncertainty graph (*LUG*). The SAG employs a labeling technique which exploits the redundant sub-structure, if any, of arbitrary sets of planning graphs.

I developed a belief-space progression planner called $POND$ to evaluate the technique. I improve the use of the *LUG* within POND by applying our SAG technique. I found an optimized form, $SLUG$, of the state agnostic version of the *LUG*, and the $\mathcal{C}$*SSAG* for the $\mathcal{M}c$*LUG*. These optimized forms improve worst-case complexity, which carries through to the experimental results.

I compared POND to state of the art planners in conformant and conditional planning. I demonstrated that, by using $SLUG$ and $\mathcal{C}$*SSAG*, POND is highly competitive with the state of the art in belief-space planning. Given the positive results in applying SAG, I see promise in applying SAG to other planning formalisms.

CHAPTER 8

MULTI-OBJECTIVE CONDITIONAL PROBABILISTIC PLANNING

Previous chapters developed a family of reachability heuristics to guide single objective search (i.e., the heuristic estimated the cost to complete a partial plan). For example, Chapter 2 followed the traditional formulation of probabilistic planning: minimize plan cost, subject to a probability of goal satisfaction threshold. This chapter breaks from tradition by recognizing that, for a number of reasons, probabilistic planning is better formulated as multi-objective search. The reasons include the ability to provide users with a Pareto set of plans (while handling the single objective formulation as a special case), the formulation of the first cost (and probability) optimizing search for conditional probabilistic planning, and the inclusion of other objectives (e.g., the number of plan branches to model limited contingency planning (Meuleau and Smith, 2003)). In this chapter, I describe a multi-objective search algorithm, called multi-objective LAO^* ($MOLAO^*$), that is used to find conditional and conformant probabilistic plans.

1. Introduction

Probabilistic planning is the inherently multi-objective problem of optimizing both plan cost and probability of goal satisfaction. However, as both Kushmerick *et al.* (1994) and Littman (1997) point out (and as I described in previous chapters), it is possible to bound one of the objectives and optimize the other with a single objective search. Bounding an objective can be useful to users that understand a domain well. However, when users do not fully understand a domain – which is precisely when they need an automated planner – they may not be able to place meaningful bounds. For example, they may (in error) place a cost bound, below which no plan can satisfy the goal with non-zero probability, or place a probability bound that can only be met by unreasonably costly plans. Instead, providing a user with a Pareto set of plans allows them to select among feasible alternatives.

Another, more technical, reason to pursue a multi-objective formulation is motivated by the difficulty of finding a solution to the single objective formulation of conditional probabilistic planning. By using the multi-objective formulation and applying bounds to the Pareto set of plans, I can easily select the plan that satisfies the goal with at least τ probability at minimal cost. The question is whether the additional overhead incurred by multi-objective optimization is justified. To see why a single formulation of conditional probabilistic planning as belief space search is perhaps ill-suited for a cost-optimizing search, consider the following.

Recall that I formulate conformant (non-observable) probabilistic planning as a forward-chaining (single objective) AO* search in the space of belief states that optimizes plan cost. If the search finds a belief state satisfying the goal with probability no less than τ, then the action sequence leading to it is a feasible plan. When it comes to conditional planning however, the situation is more complex. The probability of satisfying the goal is the expectation of satisfying it in each branch. To ensure plan feasibility (i.e., the probability of goal satisfaction exceeds τ), I must consider both (i) the probability of executing each branch and (ii) the probability that each satisfies the goal. In conformant planning (i) is always 1.0, making (ii) a belief state property. In conditional planning, feasibility is a property of the entire plan (through aggregate probabilities of branches). Naturally, during synthesis there may be many sub-plan options within the different branches. Finding a cost-optimal feasible plan involves deciding between sub-plans (in each branch) that have different costs and probabilities of goal satisfaction. Unfortunately, as the following example illustrates, it is not possible to decide which sub-plan is best in one plan branch without considering sub-plan options in the other branches. That is, I need to somehow combine these *non-dominated* options across branches.

Example:*The conditional plan π (Figure 90) starts with action a, which has two probabilistic outcomes and an execution cost of one. After the first outcome (whose probability is 0.2) it is*

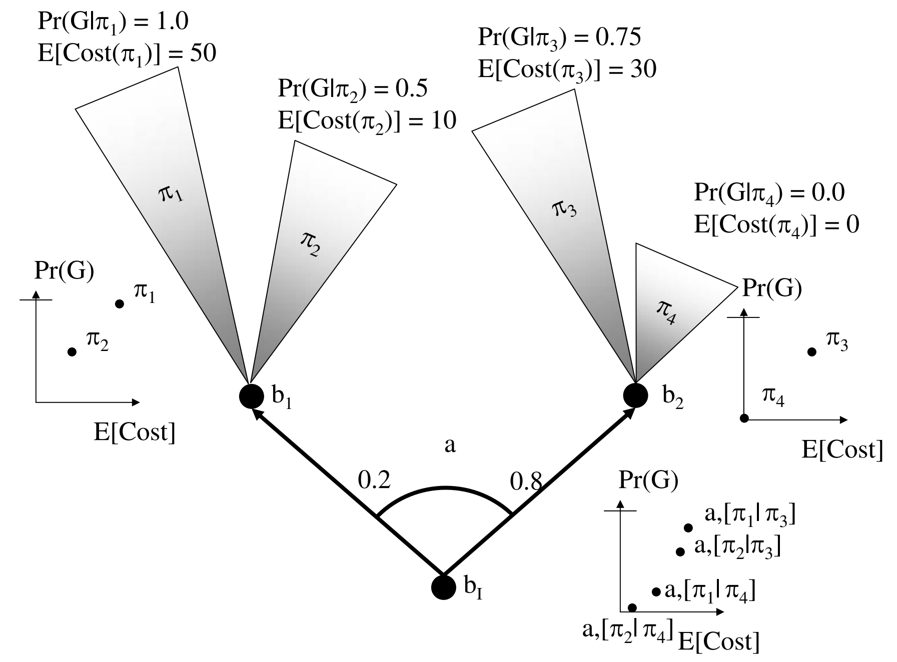


Figure 90. Example of Conditional Probabilistic Planning.

possible to execute sub-plan π_1 that achieves the goal with 1.0 probability and expected cost 50 or π_2 that achieves the goal with 0.5 probability and expected cost 10. After the second outcome (whose probability is 0.8) it is possible to execute sub-plan π_3 that achieves the goal with 0.75 probability and expected cost 30 or π_4 that achieves the goal with 0.0 probability and expected cost 0. Using the following choices for sub-plans will result in different plan costs and probability of goal satisfaction:

Plans π	E[Cost(π)]	Pr($G\|\pi$)
$a, [\pi_1\|\pi_3]$	1+(.2)50+(.8)30 = 35	(.2)1+(.8).75 = .8
$a, [\pi_2\|\pi_3]$	1+(.2)10+(.8)30 = 27	(.2).5+(.8).75 = .7
$a, [\pi_1\|\pi_4]$	1+(.2)50+(.8)0 = 11	(.2)1+(.8)0 = .2
$a, [\pi_2\|\pi_4]$	1+(.2)10+(.8)0 = 3	(.2).5+(.8)0 = .1

If $\tau = 0.6$, then using sub-plans π_2 and π_3 exceeds the threshold with a minimal expected cost of 27. Notice that if I commit to sub-plans in each branch without considering other branches, I may choose π_1 and π_3 (because each individually exceeds τ). Yet, this aggressive (and incomplete) strategy finds a feasible plan with a *higher* cost of 35. Or, in a single objective search I may pick the sub-plan with minimal cost, meaning I would choose π_2 and π_4 for a total cost of 3 and 0.1 probability of goal satisfaction (not exceeding τ).

One alternative to solving this problem is to formulate the problem as A* search in plan space. Each search node will correspond to a partial plan, and the search branches over all possible extensions of the plans. In the example above, there would be a search node corresponding to each of the four plans listed in the table above. Note that there is significant redundance among these four search nodes – each contains action a and may have one of the sub-plans π_1 to π_4 in common. A more desirable approach would capture this redundancy more compactly, omitting multiple copies of identical plan fragments. Formulating the problem as search in belief state space can avoid this redundancy, but as I noted above, single objective search in belief state space can make poor commitments in each branch (by considering only probability or only cost).

For belief space search to be effective, it must delay such pre-mature commitments in each branch. This chapter describes how this is done. Each belief state maintains a Pareto set of non-dominated sub-plans and, through multi-objective dynamic programming (Henig, 1983), communicates these options to its ancestors. The dynamic programming combines non-dominated options in child branches to define the Pareto set of sub-plans at the parent. Through a bottom-up computation, the initial belief state maintains its Pareto set of plans. If desired, I can apply a probability threshold to select the cost-optimal feasible plan. To compute these plans, I search the belief state space from the initial belief state using a multi-objective generalization of the LAO^* algorithm, called $MOLAO^*$.

At first glance, it would seem as if combining multi-objective optimization and probabilistic planning (two notoriously computationally difficult tasks) will not scale. I tame the complexity of the search, by pursuing a number of speed-ups. The most notable of these speed-ups is to use the $\mathcal{C}SSAG$ relaxed plan to guide the search. I also discuss variations on $MOLAO^*$ that focus synthesis on likely branches, reduce the complexity of dynamic programming, and reduce the size of Pareto sets by using ϵ-domination (Papadimitriou and Yannakakis, 2000).

I also take advantage of the multi-objective formulation to include other objectives for more expressive planning problems, such as limited contingency planning (LCP). My approach to LCP trades the complexity of increasing the state space, as Meuleau and Smith (2003), for the complexity of using another objective: the number of plan branches.

In the next section, I formally redefine the planning model to capture some additional notational conventions. The following section discusses two single objective dynamic programming formulations of conditional probabilistic planning and shows how they rely on very restrictive assumptions. I then show how a *novel* multi-objective formulation lifts these assumptions in the successive section. The next section describes $MOLAO^*$ and the speed-ups, followed by an empirical evaluation, a discussion of limited contingency planning, related work, and conclusions.

2. Background & Representation

I will explore two variations on the model for conditional probabilistic planning problems, where the difference is in terms of observability. I phrase the models as "flat" state-based (PO)MDPs, where the notation in this chapter supersedes the notation in Chapter 2 (however they are quite similar). The underlying model is still propositional to facilitate planning graph heuristics, but as far as search is concerned, it operates on a flat representation. In the next two sections, I will define different solution concepts for these models.

Full Observability: The fully-observable conditional probabilistic planning model (S, A, T, s_I, G) is defined as:

- S, a set of states
- A, a set of actions
- $T(s, a, s')$, a state transition relation
- s_I, an initial state
- $G(s)$, a goal state function

The state transition relation describes a probability distribution: $T(s, a, s') = Pr(s'|a, s)$. A subset of actions $A(s) \subseteq A \cup \perp$ are applicable in each state s. The $\perp$ action signifies no action is performed in s. The goal state function $G : S \rightarrow [0, 1]$ maps each state to a probability it satisfies the goal. In this model, $G(s) = 1$ or $G(s) = 0$ for every state (later we will see that $0 \leq G(s) \leq 1$ in belief state MDPs). Each state s where $G(s) = 1$ is absorbing (i.e., $A(s) = \{\perp\}$). Applying each action incurs a cost $c(a)$, which I will assume is uniform in this chapter (with the exception that $c(\perp) = 0$).

Partial Observability: The partially-observable conditional probabilistic planning model is given by $(S, A, T, b_I, G, O, \Omega)$, defined as before, and as:

- b_I, an initial belief state
- $O(s, a, o)$, an observation function
- Ω, a set of observations

A belief state $b : S \rightarrow [0, 1]$ is a probability distribution that maps each state to a probability, such that $\sum_{s \in S} b(s) = 1.0$. The set $A(b) = \cap_{s \in S : b(s) > 0} A(s)$ contains all actions applicable in a belief state b. The observation function $O(s, a, o) = Pr(o|a, s)$ characterizes a probability distribution over observations $o \in \Omega$ received upon transitioning to state s after executing action a.

I define the belief state reached by applying a in b as $b_a(s') = \sum_{s \in S} b(s)T(s, a, s')$ and the belief state b_a^o (the belief state after receiving observation o in belief state b_a) as $b_a^o(s') = \alpha O(s', a, o) b_a(s')$, where α is a normalization factor. The probability of receiving observation o in belief state b_a (i.e., the probability of reaching belief state b_a^o upon executing a in b) is defined as $T(b, a, o, b_a^o) = \alpha = \sum_{s \in S} b_a(s) O(s, a, o)$.

Fully-observable MDPs and Belief state MDPs: I am ultimately interested in solving conditional probabilistic planning problems with partial observability. However, I find it convenient to describe my approach in terms of fully-observable MDPs. To clarify the correspondence between the fully-observable and partially observable model, consider the following transformation, called a belief state MDP. A belief state MDP $(\tilde{S}, A, \tilde{T}, \tilde{s}_{b_I}, \tilde{G})$ for the POMDP $(S, A, T, b_I, G, O, \Omega)$ is given by the following equivalences, where each state $\tilde{s}_b \in \tilde{S}$ corresponds to a belief state b:

- $\tilde{T}(\tilde{s}_b, a, \tilde{s}_{b_a^o}) = T(b, a, o, b_a^o)$
- $\tilde{G}(\tilde{s}_b) = \sum_{s \in S} b(s) G(s)$

$\tilde{S}$ is an infinite set[1] of (belief) states, A is an unchanged set of actions, $\tilde{s}_{b_I}$ is the initial (belief) state, and $\tilde{G}$ is the probability each (belief) state satisfies the goal. Under this representation, $\tilde{G}(\tilde{s}_b)$ maps every (belief) state to a probability of satisfying the goal, such that $0 \leq \tilde{G}(\tilde{s}_b) \leq 1$. In the following, I remove all distinction between belief state MDPs and fully-observable MDPs, given the equivalences described above, and consider only fully-observable MDPs.

3. Single Objective Formulation

Most works that address conditional probabilistic planning optimize a single objective: probability of goal satisfaction or expected cost. In the following, I describe these approaches and characterize the Bellman optimality equation for each.

Maximizing Probability: Optimizing the probability of goal satisfaction completely ignores plan cost, except, perhaps, when breaking ties between plans that satisfy the goal with the same probability. With no concern for plan cost, it is possible for planners to search for increasingly longer plans to improve the probability of goal satisfaction. An often used solution to this problem is to assume a finite horizon of feasible plans, finding the highest probability plan with the given horizon k. The Bellman optimality equation for maximizing the probability of goal satisfaction for each state s, for each horizon $0 \leq t < k$ is:

$$\begin{aligned} J(s,t) &= \min_{a \in A(s)} q(s,t,a) \\ \pi(s,t) &= \operatorname*{argmin}_{a \in A(s)} q(s,t,a) \\ q(s,t,a) &= \sum_{s' \in S} T(s,a,s') J(s',t+1) \end{aligned}$$

and $J(s,k) = 1 - G(s)$. The objective is to compute the minimal $J(s_I, 0)$ and the corresponding plan π.

[1]While dealing with an infinite set of states is challenging, I rely on search algorithms (described later) to consider a subset of the states reachable from s_I by some sequence of actions.

In the example, each sub-plan π_1 to π_4 could have the same expected cost/length by assuming there are persistence actions. If we assume that the actions are unit cost and $k = 51$, then the optimal plan would select sub-plans π_1 and π_3 because they maximize probability of goal satisfaction.

Minimizing Expected Cost: Another solution is for planners to optimize expected cost to reach the goal. If the goal is not reachable with probability 1, then it is customary to assume a cost $c_{\not G}$ of not reaching the goal.[2] The Bellman optimality equation for maximizing expected cost for each state s is:

$$\begin{aligned}
J(s) &= \min_{a \in A(s)} q(s,a) \\
\pi(s) &= \operatorname*{argmin}_{a \in A(s)} q(s,a) \\
q(s,a) &= c(a) + \sum_{s' \in S} T(s,a,s')J(s') \\
q(s,\perp) &= c_{\not G}(1 - G(s))
\end{aligned}$$

The objective is to compute the minimal $J(s_I)$ and the corresponding plan π.

Assuming a cost for not reaching the goal indirectly influences the optimal probability of satisfying the goal, given the costs of potential plans. To see this, consider how the value for $c_{\not G}$ affects the choice between π_1 and π_2 in the example. If $c_{\not G}$ = 79, then π_1 has a total cost of (1-1)79+50 = 50 and π_2 has a total cost of (1-0.5)79+10 = 49.5, making π_2 a better choice. If $c_{\not G}$ = 81, then π_1 has a total cost of (1-1)79+50 = 50 and π_2 has a total cost of (1-1)81+10 = 50.5, making π_1 a better choice. However, if $c_{\not G}$ = 80, then π_1 has a total cost of 50 and π_2 has a total cost of 50, making the choice ambiguous. With a small variation in $c_{\not G}$ the best plan (and the probability of goal satisfaction) can change. To model a problem with a goal satisfaction probability threshold requires knowledge of the costs of alternative plans. However, if we knew the costs of alternative plans, then what is the point of using an automated planner?! Nevertheless, optimizing expected

[2] This is analogous to reward models that assign negative reward to actions and positive reward to goal states.

reward does have its merits when the probability of goal satisfaction is not of primary concern to the domain modeler. Finding an optimal plan is much more critical in this formulation because (without explicit bounds on the probability of goal satisfaction) a suboptimal plan may have a low probability of goal satisfaction despite having a relatively low cost.

4. Multi-objective Formulation

Instead of carefully engineering problems by imposing a horizon bound or a cost for failing to satisfy the goal, it is possible to directly solve the conditional probabilistic planning problem with a more natural multi-objective formulation. The formulation minimizes plan cost and maximizes probability of goal satisfaction (omitting a horizon bound or goal cost). Committing to a multi-objective formulation opens the possibility for other objectives (e.g., number of branches, makespan, resource consumption, etc.), so I present a general multi-objective formulation and point out specific details for computing the expected cost and probability of goal satisfaction objectives. In a later section, I discuss limited contingency planning by introducing the number of branches as a third objective.

I introduce an extension to the value function, such that $J(s)$ becomes a Pareto set. Each point of the Pareto set $q(s,a) \in J(s)$ represents a vector of n objectives, such that $q(s,a) = (q_0(s,a), q_1(s,a), ..., q_{n-1}(s,a))$, and $J(s) = \{q(s,a) | \neg\exists_{q'(s,a') \in J(s)} q'(s,a') \prec q(s,a)\}$ where $J(s)$ contains only non-dominated points. A point $q'(s,a')$ dominates another $q(s,a)$ if it is as good in all objectives, and strictly better in at least one:

$$\begin{aligned} q'(s,a') \prec q(s,a) \Leftrightarrow \quad & \forall_i q'_i(s,a') \leq q_i(s,a) \wedge \\ & \exists_i q'_i(s,a') < q_i(s,a) \end{aligned}$$

Each point is mapped to a best action by π (i.e., $\pi(s, q(s,a)) = a$), making it possible for two sub-plans to start with the same action but have different values.

Each $q(s,a) \in J(s)$ is computed in terms of its successors' values. Because each successor $(s', s'', ...)$ is associated with a Pareto set $(J(s'), J(s''), ...)$, it is possible to define a different $q(s,a)$ from each element $(q(s',a'), q(s'',a''), ...)$ of the cross product of the Pareto sets $(J(s') \times J(s'') \times ...)$. We saw this in the example, where it was possible to have $|J(\tilde{s}_{b_I})| = 4$ because $|J(\tilde{s}_{b_1})| = |J(\tilde{s}_{b_2})| = 2$. The cross product of $J(\tilde{s}_{b_1})$ and $J(\tilde{s}_{b_2})$ contains four elements, each used to define an element of $J(\tilde{s}_{b_I})$. The actual number of elements in a Pareto set may be less because some elements will be dominated.

I define expected cost and probability of *not* satisfying the goal objectives for probabilistic conditional planning, $q(s,a) = (q_0(s,a), q_1(s,a))$, such that:

$$\begin{aligned}
q_0(s,a) &= c(a) + \sum_{s' \in S} T(s,a,s') q'_0(s',a') \\
q_0(s,\perp) &= 0 \\
q_1(s,a) &= \sum_{s' \in S} T(s,a,s') q'_1(s',a') \\
q_1(s,\perp) &= 1 - G(s)
\end{aligned} \tag{8.1}$$

Each $q'(s',a')$ in the above equations is from an element of the cross product of successor Pareto sets.

From the example, $J(\tilde{s}_{b_I}) = \{(35, 0.2), (27, 0.3), (11, 0.8), (3, 0.9)\}$. The first element of $J(\tilde{s}_{b_I})$ is defined by $q(\tilde{s}_{b_1}, (50, 0.0)) \in J(\tilde{s}_{b_1})$ and $q(\tilde{s}_{b_2}, (30, 0.25)) \in J(\tilde{s}_{b_2})$ as $q_0(\tilde{s}_{b_I}, a) = 1 + (0.2)50 + (0.8)30 = 35$ and $q_1(\tilde{s}_{b_I}, a) = (0.2)0.0 + (0.8)0.25 = 0.2$.

To solve the single objective probabilistic conditional planning problem, I am interested in finding a plan starting with the action $a = \pi(s_I, (q_0(s_I,a), q_1(s_I,a)))$ such that the plan has minimal cost $q(s_I,a) = \min_{q(s_I,a) \in J(s_I)} q_0(s_I,a)$ subject to the goal satisfaction threshold $\tau \leq 1 - q_1(s_I,a)$. The remaining actions in the plan are those actions used to define each $q(s,a)$ for the actions in the plan. For example, if the plan starts by executing a and it results in a single state s', then the action to execute in s' is a' where $q'(s',a')$ was used to compute $q(s_I,a)$.

Computing the Multi-objective Bellman Equation: It can take $O(|A(s)||J(s')|^{|S|})$ time to compute the multi-objective Bellman equation *once* for a state s because there are $|A(s)|$ actions are applicable in s, each results in at most $|S|$ successor states s', and there are $|J(s')|$ Pareto optimal sub-plans for each state s'. In a later section, I identify ways to reduce this complexity by limiting the size of the Pareto sets $J(s)$.

5. Multi-objective LAO^*

Hansen and Zilberstein (2001) introduce Looping AO* (LAO^*) search as a technique for solving stochastic shortest path and MDP problems. Unlike the traditional value iteration and policy iteration techniques for solving MDP problems, LAO^* generalizes AO* search (Nilsson, 1980) to handle loops in the search graph. The idea is to expand a limited region of the state space, over which value iteration is used to identify a partial plan. In each iteration, LAO^* expands unexpanded fringe states of the partial plan, and performs value iteration. If the fringe state values are initialized with an admissible heuristic, LAO^* terminates when it identifies an optimal solution. LAO^* termination relies on two criterion: the difference between upper and lower bounds on the value function is below a threshold, and the solution contains no unexpanded states. In my setting, solutions are plans and not policies, meaning every plan path eventually ends in a terminal state. By labeling terminal states as “solved” (as done in AO*), we can propagate solved labels to identify if a solution contains unexpanded states.

The main generalization needed for a multi-objective LAO^* ($MOLAO^*$) is to reinterpret the J-values as J-sets and compute them as such. This also means that there may be several non-dominated partial solutions that $MOLAO^*$ can expand each iteration. Figures 91 and 92 describe the $MOLAO^*$ search algorithm using J-sets to find a Pareto set of conditional plans. The $MOLAO^*$ function in Figure 91 contains the outer loop that calls `ExpandPlan` to compute a set

```
$MOLAO^*$()
1: $i = 0$
2: **repeat**
3:    Z = ExpandPlan($s_I, i^{++}$)
4:    AddAncestors(Z)
5:    VI(Z)
6: **until** (ConvergenceTest($s_I$) == T)

ExpandPlan($s, i$)
1: **if** expanded($s$) $< i$ **then**
2:    Z = Z $\cup s$
3:    **if** expanded($s$) == 0 **then**
4:       expanded($s$) = $i$
5:       ExpandState($s$)
6:    **else**
7:       expanded($s$) = $i$
8:       **for** $q(s, a) \in J(s)$s.t. $\neg$solved$(q(s, a))$ **do**
9:          **for** $s' \in S : T(s, a, s') > 0$ **do**
10:            Z' = ExpandPlan($s', i$)
11:            Z = Z $\cup$ Z'
12:         **end for**
13:      **end for**
14:   **end if**
15: **end if**
16: return Z
```

Figure 91. $MOLAO^*$ Search Algorithm.

of states Z that are reachable by the set of non-dominated partial plans. To Z, AddAncestors adds all states that can reach a state $s \in$ Z with a non-dominated plan. The set of partial plans is revised by calling VI(Z) to update $J(s)$ for each $s \in$ Z. When ConvergenceTest indicates that $J(s_I)$ has converged (i.e, all non-dominated solutions are labeled solved, and upper and lower bounds on their values is below a threshold), it is possible to stop.

The ExpandPlan function recursively traces each non-dominated partial plan (lines 7-13) to find leaf states to expand (lines 4-5). The ExpandState function applies each action $a \in A(s)$ to generate successor states. Each successor state s' has its Pareto set $J(s')$ initialized and expanded(s') is set equal to zero. Initializing $J(s')$ involves adding a point $q(s', \perp)$ where

```
Backup($s$)
1: $J'(s) = \emptyset$
2: for $a \in A(s)$ do
3:     $W = \bigtimes_{s':T(s,a,s')>0} J(s')$
4:     for $w \in W$ do
5:         Compute $q(s,a)$ with each $q'(s',a') \in w$
6:         solved$(q(s,a)) = \bigwedge_{q'(s',a') \in w}$ solved$(q'(s',a'))$
7:         UpdateDominance($q(s,a), J'(s)$)
8:     end for
9: end for
10: error = ComputeError($J'(s), J(s)$)
11: $J(s) = J'(s)$
12: return error
```

Figure 92. $MOLAO^*$ state value Backup.

solved($q(s', \perp)$) is true, indicating that it is possible to end the plan at s'. I can also add heuristic points $q(s, *)$ to $J(s)$ that indicate heuristic estimates of non-dominated sub-plans. Each heuristic point is marked unsolved. Through dynamic programming, each $J(s)$ will contain some heuristic points that indicate the value of partial plans rooted at s. Later, I discuss which and how many heuristic points I use.

The VI function performs value iteration on a set of states (i.e., iteratively calling Backup, Figure 92, for each state until the maximum change in some $J(s)$ falls below a threshold). The Backup function computes $J(s)$ by applying every action in the loop (lines 2-9). For each action, W is the cross product of Pareto sets of successor states (line 3). For each element $w \in W$, a point $q(s,a)$ is computed (as in Equation 8.1) with the points $q'(s',a') \in w$ (line 5). Each new point is marked solved if each of the successor points is solved (line 6). The UpdateDominance function (line 7) maintains the Pareto set by checking if each new point is dominated, or dominates points already in the Pareto set.

I deliberatively leave ConvergenceTest and ComputeError undefined because I will pursue a variation on $MOLAO^*$ that I use to compute *feasible* conditional probabilistic plans quickly. Many of the speed-ups involved will affect completeness and admissibility.

5.1. $MOLAO^*$ **Speedups.** In the following, I describe four improvements to $MOLAO^*$ that I use to find feasible, but suboptimal, solutions quickly. The first is a modified version of $MOLAO^*$, named m$MOLAO^*$. The second describes heuristics. The third involves Pareto set approximations. The fourth simulates the current partial plan to focus synthesis, similar to RTDP (Barto *et al.*, 1995).

m$MOLAO^*$**:** The variation of $MOLAO^*$ that I pursue is very similar to the "efficient" version of LAO^* presented by Hansen and Zilberstein (2001) (c.f. Table 7). The idea is to combine value iteration with solution expansion by calling the `Backup` function for each state after recursively calling `ExpandPlan` for each of child state reached by an action in $J(s)$. Figure 93 describes my version of the $MOLAO^*$ algorithm that not only combines value iteration with solution expansion, but also stops when a *feasible* (labeled solved) solution satisfies the goal with probability no less than τ. I omit the convergence test used in LAO*, so that I can stop at the first feasible plan.

$MOLAO^*$ **Heuristics:** Adding heuristic points increases the size of J-sets, which I previously noted as a major source of complexity in computing state backups. There are two techniques I use to mitigate the increase in J-set size. First, I use a single heuristic point that estimates the cost to satisfy the goal with probability 1. Second, I limit how solved and not solved points are combined in the state backups. Specifically, I only combine solved points with solved points, and not solved points with not solved points. Formally, I replace W in line 3 of `Backup` with W':

$$W' = W \backslash \{w | q(s,a), q'(s',a') \in w, w \in W,$$
$$\text{solved}(q(s,a)) \neq \text{solved}(q'(s',a'))\}$$

If I let $J_{sol}(s)$ denote all solved points and $J_h(s)$ denote all not solved (heuristic) points for a state s, then restricting the combination of solved and not solved points will bring the complexity of computing the multi-objective Bellman equation from $O(|A(s)|(|J_{sol}(s)| + |J_h(s)|)^{|S|})$

```
m$MOLAO^*$()
1: $i = 0$
2: repeat
3:   mExpandPlan($s_I, i^{++}$)
4: until $\exists_{q(s_I,a) \in J(s_I)}$feasible$(q(s_I, a))$

mExpandPlan($s, i$)
1: if expanded($s$) $< i$ then
2:   if expanded($s$) == 0 then
3:     expanded($s$) = $i$
4:     ExpandState($s$)
5:   else
6:     expanded($s$) = $i$
7:     for $q(s, a) \in J(s)$ s.t. $\neg$solved$(q(s, a))$ do
8:       for $s' \in S : T(s, a, s') > 0$ do
9:         mExpandPlan($s', i$)
10:      end for
11:    end for
12:  end if
13:  Backup(s)
14: end if
```

Figure 93. Modified $MOLAO^*$ Search Algorithm.

to $O(|A(s)|(|J_{sol}(s)|^{|S|} + |J_h(s)|^{|S|}))$. Notice that because I use a single heuristic point per Pareto set and combine points in this fashion, there will only be a single partial plan expanded in line 7 of mExpandPlan (i.e., $\forall s|J_h(s)| = 1$).

The specific heuristic that I use is extracted from the $\mathcal{C}SSAG$. It estimates the cost to satisfy the goal with at least a given probability (which I set to 1.0) by computing a relaxed plan. The $\mathcal{C}SSAG$ relaxed plan estimates the conformant probabilistic plan cost, which can be seen as an additional relaxation that ignores observations. I also compare with a non-heuristic strategy where I set the cost to reach the goal with 1.0 probability to zero.

Approximating Pareto Sets: As with most multi-objective optimization problems, a Pareto set can contain an exponential number of non-dominated solutions. There exists a range of techniques for computing these Pareto sets and I explore one idea to reduce their size. It relies on the notion of

ϵ-domination (Papadimitriou and Yannakakis, 2000) to prune solutions. By inflating each objective of other sub-plans by a factor of $1 + \epsilon$, it is possible to approximate the Pareto to within a relative error of ϵ in each objective by using a different definition of domination:

$$q'(s,a') \prec^{\epsilon} q(s,a) \Leftrightarrow \forall_i q'_i(s,a') \leq (1+\epsilon)q_i(s,a)$$

The resulting Pareto set is polynomial sized in the number of sub-plans and $\frac{1}{\epsilon}$ (c.f. Theorem 1, Papadimitriou and Yannakakis, 2000), but may still take exponential time to compute. However, because the number of non-dominated sub-plans tends to increase during bottom-up dynamic programming (i.e., a parent's Pareto set is exponential in its childrens' Pareto sets) pruning more sub-plans can have a large effect.

Simulated Expansions: Instead of expanding every leaf state of the current partial plan, it is possible to expand a single leaf state that is reached by simulating the partial plan. This variation of $MOLAO^*$ called $MOLAO^{*sim}$ samples a single successor state s' from the transition relation in line 8 of `mExpandPlan` instead of iterating over every successor. The intuition for this variation is to concentrate plan synthesis on the most likely plan branches.

6. Empirical Results

I implemented $MOLAO^*$ within $POND$ to take advantage of its reachability heuristics computed from the $\mathcal{CSSAG}$. In the following, each reference to $MOLAO^*$ is with respect to the modified version (described above). Given my description of approaches for solving conditional probabilistic planning, it seems that no previous work is directly relevant for empirical comparison. As a coarse comparison, I compare with the Zander planner (Majercik and Littman, 2003) on the single objective conditional probabilistic planning problem in the following manner. Zander finds the maximum probability k-horizon plan with a stochastic satisfiability solver. By increasing k incrementally, it is possible to find the optimal probability of goal satisfaction τ for different expected plan lengths. Unlike my approach, the conditional plans generated by Zander require each plan branch is length k. Thus, for the same τ, $POND$ may find lower expected length plans by virtue of removing this structural assumption. In fact, $POND$ generates conditional plans as digraphs, where paths may have heterogenous lengths.

The domains that I use for comparison with Zander include Coffee Robot, GO-5, and Ship-Reject from Majercik and Littman (2003). Noticing that these domains are small (in the number of states and actions, and the plan lengths), I developed two domains: First-Responders and Grid. Both GO-5 and First-Responders plans use loops in the belief state space. The following table lists the number of Actions $|A|$, Propositions $|P|$, and observations $|O|$ in each problem.

Problem	$\|A\|$	$\|P\|$	$\|O\|$
Coffee Robot	6	8	4
GO-5	5	6	5
Ship-Reject	4	5	2
FR1	82	29	22
FR2	116	45	28
FR3	200	54	42
Grid	5	20	2

The First-Responders domain is inspired by the planning required of dispatchers in disaster scenarios. There may be multiple victims (of unknown health) that need to be treated on the scene (with a lower probability of improvement) or taken to the hospital by an ambulance (where their health will improve with high probability). There may also be fires at several locations that fire trucks can extinguish. Both fire and ambulances can report what they observe at a location. The goal is to bring each victim to health and extinguish all fires. We use three instances: FR1 has two fires, two victims, and four locations; FR2 has two fires, four victims, and four locations; and FR3 has two fires, two victims, and nine locations.

The Grid domain is an adaptation of the grid problem first described by Hyafil and Bacchus

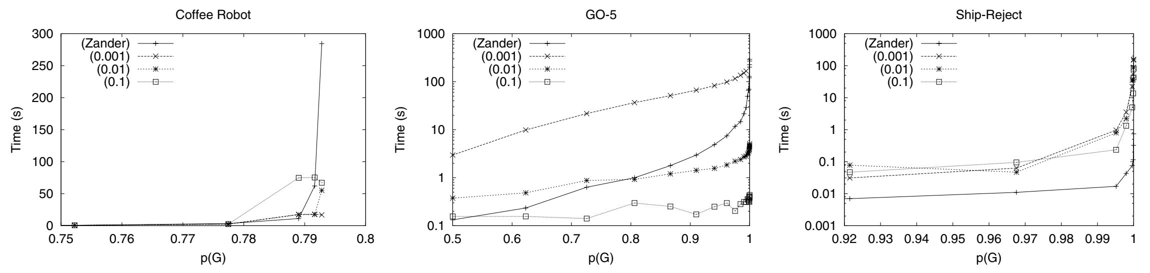


Figure 94. Run times (s) vs. τ for Coffee Robot, GO-5, and Ship-Reject.

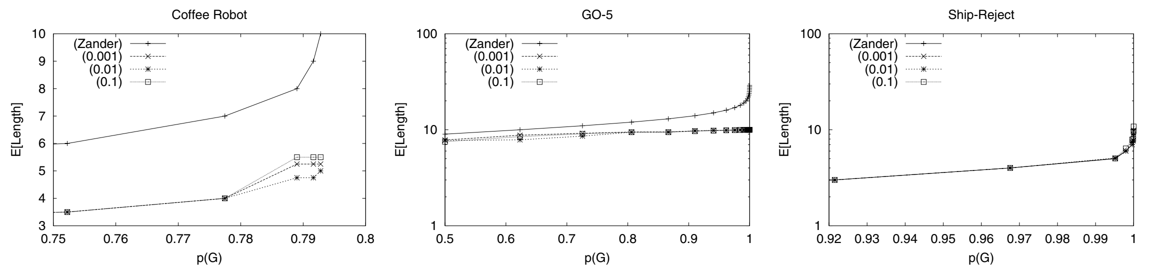


Figure 95. Expected Plan Cost vs. τ for Coffee Robot, GO-5, and Ship-Reject.

(2004) to include an action with observations. Like the original domain, a robot is moving about a 10x10 grid. The robot starts in the center of the grid and must reach a given corner. Its actions move it in the intended direction with 0.8 probability and in an adjacent direction with 0.1 probability. A sensory action allows the robot to perfectly sense the presence of a wall.

Zander Domains: The plots in Figures 94 and 95 show results for the Coffee-Robot, GO-5, and Ship-Reject domains. I compare Zander with $MOLAO^*$ without a heuristic or randomized expansions by varying the value of ϵ for approximate Pareto sets. Using the $\mathcal{M}cLUG$ heuristic in these problems showed little improvement. The plot legends indicate the results for $POND$ by the value of ϵ. The total planning time in seconds is shown in Figure 94 and the expected plan length is shown in Figure 95 for several values of τ. Despite $POND$ not guaranteeing optimal plans, it finds plans that are less than or equal to the expected length of plans found by Zander (per the discussion above). In terms of planning time, in most cases $POND$ performs better than Zander as ϵ increases

		$MOLAO^{*sim}$					$MOLAO^{*h}$					$MOLAO^{*h-sim}$				
	τ	T(s)	E[C]	PS	DS	TS	T(s)	E[C]	PS	DS	TS	T(s)	E[C]	PS	DS	TS
FR1	0.25	314.8	9.8	101.0	3.0	19.0	41.0	12.1	165.2	14.8	20.2	14.5	12.0	42.6	1.2	13.8
	0.50	352.7	19.1	185.0	13.0	24.0	56.8	21.9	335.2	31.8	31.0	30.9	22.5	85.8	4.4	17.0
	0.75	370.2	25.6	277.0	25.0	27.0	62.1	26.8	448.2	47.2	37.4	42.4	27.5	160.2	10.0	20.8
	0.95	-	-	-	-	-	81.7	30.1	533.6	65.8	31.8	145.8	29.0	330.8	29.5	30.8
FR2	0.25	-	-	-	-	-	291.9	15.5	447.2	65.4	39.0	160.3	18.0	86.6	2.6	17.2
	0.50	-	-	-	-	-	318.6	26.3	469.4	53.4	31.4	130.9	24.7	115.4	5.6	19.6
	0.75	-	-	-	-	-	370.1	32.8	585.4	77.8	37.0	231.9	31.1	218.6	17.4	25.4
	0.95	-	-	-	-	-	457.4	39.3	816.8	98.8	41.2	292.2	37.5	373.0	35.0	30.0
FR3	0.25	-	-	-	-	-	420.9	12.8	392.0	34.2	33.0	252.8	15.0	69.0	1.8	15.2
	0.50	-	-	-	-	-	468.2	23.5	553.3	50.0	39.3	265.0	27.0	141.6	4.4	20.2
	0.75	-	-	-	-	-	623.7	40.67	741.0	51.0	45.0	459.2	35.3	248.0	13.0	26.4
	0.95	-	-	-	-	-	624.6	39.2	865.0	70.0	49.0	-	-	-	-	-
Grid	0.25	-	-	-	-	-	32.3	10.8	21.2	0.0	6.8	21.7	13.2	19.8	0.0	6.8
	0.50	-	-	-	-	-	356.7	19.4	153.2	0.2	36.8	55.5	19.0	54.8	0.2	14.8
	0.75	-	-	-	-	-	954.5	23.6	537.6	0.8	104.8	161.3	21.5	150.6	0.0	29.8

Table 8. Results using $MOLAO^{*sim}$, $MOLAO^{*h}$, and $MOLAO^{*h-sim}$.

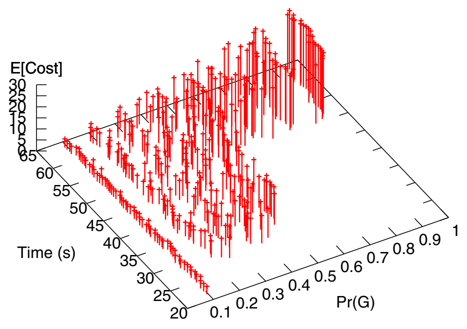


Figure 96. E[C] vs. τ over time on FR1, using $MOLAO^{*h}$.

(decreasing the size of J-sets). Zander performs better in the Ship-Reject; the $POND$ and Zander plan costs are close, indicating that Zander is better when are balanced trees. $POND$ is able to capitalize on the plans that are unbalanced trees in other domains.

New Domains: Results for the First-Responders and Grid domain are shown in Table 8. All the results are the average of 5 runs. I compare three versions of the modified $MOLAO^*$ where the superscript "sim" corresponds to using simulated expansions, and the superscript "h" corresponds to using the relaxed plan heuristic (instead of zero) for heuristic points. In each version I use $\epsilon = 0.1$ and 10 samples within each $\mathcal{CSSAG}$. I do not show results for the version of modified $MOLAO^*$, used in the Zander domains, because it was not able to solve any of the instances for any value of τ. Due to complications with encoding the domains in Zander's input format, I do not report Zander results. I expect Zander to perform comparable to $MOLAO^*$ without heuristics. The table reports total time in seconds "T(s)", plan cost "E[C]", number of unique belief states in the plan where an action is executed "PS", number of belief states with more than one parent in the plan "DS", and the number of terminal belief states "TS".

I see that the probabilistic conformant planning graph heuristic does indeed improve scalability, as I hoped. $MOLAO^{*sim}$ is only able to solve the smallest FR1 to a 0.75 probability of goal satisfaction. However, using the heuristic in $MOLAO^{*h}$ and $MOLAO^{*h-sim}$ allows $POND$ to solve every instance to a high level of probability. While the expected cost is slightly better without using the heuristic, I see the benefit of using an inadmissible, but informative heuristic. In the instances requiring the heuristic, the number of plan belief states (PS) is quite large, testifying to the scale of the plans. The number of plan belief states with more than one parent (DS) shows the benefit of using $MOLAO^*$ to find more compact digraph plans. The number of terminal belief states (TS) attests to the number of branches in the plans (modulo the number of collapsed branches, captured by DS).

While I presented results for solving the single objective conditional probabilistic planning problem, my main interest is in finding Pareto sets of plans. Figure 96 shows the Pareto set for b_I over time (after each iteration in `ExpandPlan`) in the FR1 domain while using $MOLAO^{*h}$. Each line represents a solution satisfying the goal with the indicated probability and cost. The set of feasible plans steadily increase the probability of goal satisfaction over time, suggesting how $MOLAO^*$ can be used in an anytime fashion to find a Pareto set of plans. I cut the search off once a solution is found that exceeds 0.95 probability of goal satisfaction. The search can, however, continue to find solutions with higher probability of goal satisfaction, or lower cost solutions for other probabilities of goal satisfaction.

7. Limited Contingency Planning

Limited contingency planning involves bounding the number of plan branches k to make plans more human-understandable and compact (Meuleau and Smith, 2003). Meuleau and Smith formulate k-contingency problems as a POMDP where they embed the number of contingencies as a state feature and make k copies of the state space. Executing a sensory action transitions the state to a logically equivalent state where there are fewer available contingencies. A sensory action cannot be executed in states where it introduces more than the available number of contingencies.

The grid problem, which has 100 reachable states (2^{20} total states), could be encoded as a POMDP with $100(k+1)$ states. However, this problem proved difficult for a POMDP solver when $k = 1$ (Hyafil and Bacchus, 2004), making it doubtful that it could scale with k.

In $MOLAO^*$, I can trade this state space explosion for a third objective $q_2(s,a)$. The objective is computed as $q_2(s,a) = \sum_{s' \in S: T(s,a,s')>0} q'_2(s', a')$ and $q_2(s, \perp) = 1$. I can identify plans with k or less branches when $q_2(s_I, a) \leq k$. Furthermore, I can prune a point $q(s,a)$ at any time if $q_2(s,a) > k$ because its sub-plan already exceeds the number of contingencies. I initialize all heuristic points as $q_2(s,*) = 1$ because all plans have at least one contingency. While the number of contingencies can be seen as an objective, I assume that all numbers of contingencies less than k are equally preferred and use the objective as a hard constraint for plan feasibility rather than optimization.

	τ = 0.25			τ = 0.5			τ = 0.75		
k	T(s)	E[C]	TS	T(s)	E[C]	TS	T(s)	E[C]	TS
2^0	10.4	10.7	1.0	11.9	13.7	1.0	14.1	18.0	1.0
2^1	16.2	10.9	1.5	21.8	15.0	1.2	25.8	20.5	1.0
2^2	20.9	11.1	1.5	26.5	13.6	2.0	46.5	19.4	1.7
2^3	19.1	11.1	3.0	35.3	14.7	3.7	73.0	20.6	3.0
2^6	43.8	16.3	2.8	77.8	23.4	10.6	110.5	24.9	9.8
2^9	56.4	17.0	3.8	47.7	23.2	5.4	245.5	29.1	43.0
2^{12}	52.7	18.7	4.4	69.2	27.1	9.8	146.9	30.8	23.6

Table 9. Limited contingency $MOLAO^{*h}$ on Grid.

Table 9 presents results from the Grid domain, where I vary the maximum number of branches k allowable in a feasible plan. $MOLAO^{*h}$ scales reasonably well with k. Using k as a pruning condition greatly improves time to find a plan compared to the unbounded case.

8. Related Work

My formulation of conditional probabilistic planning is based on existing work techniques for vector-valued MDPs (Henig, 1983). I define cost, probability of goal satisfaction, and number of branches as values optimized in the MDP. Where previous work concentrates on value iteration, we extend the more appropriate LAO^* algorithm to deal with multiple objectives.

As mentioned earlier, there are several techniques for solving conditional probabilistic planning. Earliest work (Draper *et al.*, 1994; Onder, 1999) used partial order planning techniques to find feasible plans (ignoring plan cost). In parallel, work on single objective MDPs and POMDPs (surveyed by Boutilier *et al.* (1999)) were developed to find cost optimal plans (using penalties for not satisfying the goal). More recently, Majercik and Littman (2003) formulated the problem as stochastic satisfiability to find the optimal probability of goal satisfaction for finite horizon plans. None of the previous works use reachability heuristics to guide plan synthesis and hence have concentrated on relatively small problems.

9. Conclusion

I have shown that embracing the multi-objective nature of probabilistic planning allows one to overcome restrictive assumptions made in single objective formulations. It also provides a richer notion of conditional plans: providing a Pareto set of plans (and sub-plans) to the plan executor introduces additional “choice” contingencies to the set of nature selected “chance” contingencies. These benefits incur a greater combinatorial overhead, but are offset by ϵ-domination techniques for Pareto set approximation. Further, because my formulation uses cost as an optimization criterion, I can make use of effective planning graph reachability heuristics that estimate plan cost. I have also shown that additional objectives, such as the number of plan branches in limited contingency planning, are easily integrated within my framework.

CHAPTER 9

CONCLUSION

Over the previous chapters, I provided a range of planning graph reachability heuristics for solving non-deterministic and probabilistic, conformant and conditional, planning problems. I also developed a new search algorithm for solving these problems. In this chapter I provide a summary, as well as a perspective on how this can be further extended.

1. Summary

With the intent of scaling planning under uncertainty, this work has made several improvements to heuristic search, both in terms of heuristics and search algorithms. The contributions include:

- **Distance Metrics For Belief States:** By defining several metrics for belief state distance and providing several planning graph heuristics to estimate the metrics, this work provided a framework for heuristic search in belief state space.

- **Symbolic Representation of Sets of Planning Graphs:** This work developed the labeled planning graph (*LUG*), which captures sets of planning graphs symbolically by using propositional formulas to label planning graph vertices.

- **Cost-Sensitive Belief State Heuristics:** The *CLUG* propagates cost information on the *LUG* to provide heuristics that reason with actions with non-uniform costs.

- **Monte Carlo Planning Graphs:** Within the $\mathcal{M}cLUG$, I developed a new relaxation for probabilistic planning that uses Monte Carlo simulation within planning graphs to deal with the uncertain outcomes of actions.

- **State Agnostic Planning Graphs:** By developing the SAG framework for planning graphs, I showed how to capture arbitrary sets of planning graphs, which had a large impact in improving heuristic computation in planning under uncertainty.

- **Heuristic Search for Conditional Probabilistic Plans:** With a desire to apply planning graph heuristics to conditional probabilistic planning I developed the $MOLAO^*$ search algorithm, which finds Pareto sets of conditional plans.

With all of these contributions the scalability of planning under uncertainty has improved considerably. The problems solved with these techniques are comparable to the size of classical planning benchmarks. As classical planning techniques continue to improve, there will be further opportunities to adapt them to consider uncertainty. In the next section, I identify several ways in which the methods described by this work can both be applied and extended.

2. Future Directions

There are a number of ways that this work can extend to new and interesting scenarios, such as integrated planning and execution, incomplete domain models, and finding sets of diverse plans. It may be also possible to improve the heuristics by incorporating observations and extend them to decision theoretic planning.

2.1. Planning and Execution. In Chapter 1, I mentioned that there are two ways to deal with uncertainty, either be reactive or proactive. In reality there are a number of hybrid strategies that incorporate aspects of both. Systems that integrate the two are likely to generate plan branches for the most critical contingencies, where it may be impossible or very costly to re-plan. In other situations, which are highly unlikely or easy for re-planning, the system would wait to re-plan when necessary. The distinction between these cases is not always obvious, unless the system actually tries to plan for them. Yet, if the system succeeds in planning the contingencies, then it does not need to worry about re-planning. Instead of using search to evaluate whether a contingency should be planned, one could use planning graph reachability heuristics to estimate the planning effort and impact of not pre-planning.

Another consideration for such systems is the amount of time it will take to plan certain branches. A system that plans and executes may have finite time before it must start the next action. Knowing how long it will take to plan can be useful in selecting which branches to pre-plan. Again, planning graph heuristics may be useful to estimate the size of the search tree within a branch, and hence estimate the time to plan.

2.2. Incomplete Domain Models. In addition to the type of incompleteness discussed in this work, which is called incomplete state information, there may also be incomplete information about action descriptions (Chang and Amir, 2006). In such scenarios the belief state not only captures uncertainty about the state of the world, but also uncertainty about preconditions and effects.

It should be possible to incorporate this type of incompleteness within the *LUG*. With this information, heuristics can guide search to trade-off exploration and exploitation. That is, exploration involves reducing the size of the belief state by trying actions to learn their descriptions, and exploitation is choosing actions that will lead to goal achievement.

2.3. Diverse Plans. Often times users desire not just a single plan, but several qualitatively different plans. Chapter 8 described a multi-objective search algorithm that finds Pareto sets of plans. Each plan in the Pareto set is different because it optimizes the planning objectives differently. Comparing two (partial) Pareto sets of the same size, the Pareto set that has relatively clustered points can be thought of as less diverse. It should be possible to guide the search for a Pareto set of plans toward plans are not clustered. A number of interesting objectives could be useful for describing the plans, such as specific partial orderings of actions, state sequences, makespan, resource utilization, and human impact.

2.4. Incorporating Observations in Reachability Heuristics. I was able to use conformant planning heuristics to scale conditional plan synthesis by ignoring observations. It may be possible to incorporate observations to improve the heuristics.

With observations, a conditional plan can disambiguate some of the possibilities characterized by the uncertainty and divide the cases among several plan branches. A conditional plan may contain many of the same actions as in a conformant plan, but divide the actions among plan branches. Therefore, it seems reasonable to use conformant relaxed plans, but post-process them to include observations. The resulting conditional relaxed plan would have an expected cost that is used as the heuristic.

2.5. Decision Theoretic Heuristics. Chapter 8 described several approaches for conditional probabilistic plan synthesis. One was based on a single-objective formulation that optimizes expected plan cost and assigning a cost penalty to not satisfying the goal. This formulation proves less desirable when the probability of goal satisfaction is bounded. However, in cases where the probability of goal satisfaction is not bounded, such a formulation can be used for decision theoretic planning. It should be possible to extend the probabilistic planning heuristics to decision theoretic planning by combining the $\mathcal{M}cLUG$ techniques with heuristics developed for partial satisfaction planning (an extension of classical planning that associates costs with actions and utilities to goals) (van den Briel *et al.*, 2004). Such probabilistic partial satisfaction heuristics would estimate net-benefit (the expected utility minus expected cost) of plan suffixes by choosing not only which goal states to achieve, but also the probability of achieving them.

REFERENCES

S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, February 2002.

A. G. Barto, S.J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, January 1995.

Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, and Jennifer Widom. Uldbs: Databases with uncertainty and lineage. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim, editors, *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 953–964. ACM, 2006.

Piergiorgio Bertoli and Alessandro Cimatti. Improving heuristics for planning as search in belief space. In Ghallab et al. [2002], pages 143–152.

Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 473–478. Morgan Kaufmann, 2001.

Susanne Biundo and Maria Fox, editors. *Recent Advances in AI Planning, 5th European Conference on Planning, ECP'99, Durham, UK, September 8-10, 1999, Proceedings*, volume 1809 of *Lecture Notes in Computer Science*. Springer, 2000.

Susanne Biundo, Karen L. Myers, and Kanna Rajan, editors. *Proceedings of the Fifteenth Inter-*

national Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA. AAAI, 2005.

Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal, Québec, Canada, August 20-25 1995*, pages 1636–1642. Morgan Kaufmann, 1995.

Avrim Blum and John Langford. Probabilistic planning in the graphplan framework. In Biundo and Fox [2000], pages 319–332.

Mark S. Boddy, Johnathan Gohde, Thomas Haigh, and Steven A. Harp. Course of action generation for cyber security using classical planning. In Biundo et al. [2005], pages 12–21.

Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In Biundo and Fox [2000], pages 360–372.

Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In Steve Chien, Subbarao Kambhampati, and Craig A. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, April 14-17, 2000, Breckenridge, CO, USA*, pages 52–61. AAAI, 2000.

Blai Bonet and Hector Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In Giunchiglia et al. [2003], pages 12–31.

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

Ronen I. Brafman and Jörg Hoffmann. Conformant planning via heuristic forward search: A new approach. In Zilberstein et al. [2004], pages 355–364.

R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

Daniel Bryce and Subbarao Kambhampati. Heuristic guidance measures for conformant planning. In Zilberstein et al. [2004], pages 365–375.

D. Bryce, S. Kambhampati, and D.E. Smith. Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research*, 26:35–99, 2006.

Anthony Cassandra, Michael Littman, and Nevin Zhang. Incremental pruning: A simple, fast, exact method for partially observable markov. In *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 54–61, San Francisco, CA, 1997. Morgan Kaufmann.

Claudio Castellini, Enrico Giunchiglia, and Armando Tacchella. Improvements to sat-based conformant planning. In Susanne Biundo and Maria Fox, editors, *6th European Conference on Planning, ECP'01, Toledo, Spain, September 12-14, 2001*. Springer, 2001.

Allen Chang and Eyal Amir. Goal achievement in partially known, partially observable domains. In Long and Smith [2006].

T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

William Cushing and Daniel Bryce. State agnostic planning graphs and the application to belief-space planning. In Veloso and Kambhampati [2005], pages 1131–1138.

J. de Kleer. An Assumption-Based TMS. *Artificial Intelligence*, 28(2):127–162, 1986.

M.B. Do and S. Kambhampati. Sapa: A scalable multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, 20:155–194, 2003.

A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, New York, New York, 2001.

Brian Drabble, editor. *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems, Edinburgh, Scotland, May 29-31, 1996.* AAAI, 1996.

Denise Draper, Steve Hanks, and Daniel S. Weld. Probabilistic planning with information gathering and contingent execution. In Hammond [1994], pages 31–36.

Thomas Eiter and Thomas Lukasiewicz. Probabilistic reasoning about actions in nonmonotonic causal theories. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 192–19, San Francisco, CA, 2003. Morgan Kaufmann.

Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In D. C. Cooper, editor, *Proceedings of the 2nd International Joint Conference on Artificial Intelligence. London, UK, September 1971*, pages 608–620. William Kaufmann, 1971.

Michael R. Genesereth and Illah R. Nourbakhsh. Time-saving tips for problem solving with incomplete information. In Richard Fikes and Wendy Lehnert, editors, *AAAI*, pages 724–730. AAAI Press/The MIT Press, 1993.

A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, 20:239–290, 2003.

Malik Ghallab and Hervé Laruelle. Representation and control in ixtet, a temporal planner. In Hammond [1994], pages 61–67.

Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors. *Proceedings of the Sixth Inter-*

national Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France. AAAI, 2002.

Enrico Giunchiglia, Nicola Muscettola, and Dana S. Nau, editors. *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*. AAAI, 2003.

Robert Goldman and Mark Boddy. Epsilon-safe planning. In *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 253–26, San Francisco, CA, 1994. Morgan Kaufmann.

Robert P. Goldman and Mark S. Boddy. Conditional linear planning. In Hammond [1994], pages 80–85.

Kristian J. Hammond, editor. *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, June 13-15, 1994, University of Chicago, Chicago, Illinois.* AAAI, 1994.

E.A. Hansen and S. Zilberstein. LAO: A heuristic-search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1–2):35–62, 2001.

M.I. Henig. Vector-valued dynamic programming. *Siam J. Cont.*, 21:490–499, 1983.

Jörg Hoffmann and Ronen I. Brafman. Contingent planning via heuristic forward search with implicit belief states. In Biundo et al. [2005], pages 71–80.

J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

J. Huang. Combining knowledge compilation and search for efficient conformant probabilistic planning. In Long and Smith [2006], pages 253–262.

Nathanael Hyafil and Fahiem Bacchus. Conformant probabilistic planning via csps. In Giunchiglia et al. [2003], pages 205–214.

Nathanael Hyafil and Fahiem Bacchus. Utilizing structured representations and csp's in conformant probabilistic planning. In Ramon López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 1033–1034. IOS Press, 2004.

Subbarao Kambhampati, Laurie H. Ihrig, and Biplav Srivastava. A candidate set based analysis of subgoal interactions in conjunctive goal planning. In Drabble [1996], pages 125–133.

Subbarao Kambhampati, Eric Parker, and Eric Lambrecht. Understanding and extending graphplan. In Steel and Alami [1997], pages 260–272.

Henry A. Kautz, David A. McAllester, and Bart Selman. Encoding plans in propositional logic. In Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996.*, pages 374–384. Morgan Kaufmann, 1996.

Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos. Extending planning graphs to an adl subset. In Steel and Alami [1997], pages 273–285.

Sven Koenig and Yaxin Liu. Sensor planning with non-linear utility functions. In Biundo and Fox [2000], pages 265–277.

James Kurien, P. Pandurang Nayak, and David E. Smith. Fragment-based conformant planning. In Ghallab et al. [2002], pages 153–162.

Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An algorithm for probabilistic least-commitment planning. In Barbara Hayes-Roth and Richard Korf, editors, *AAAI*, pages 1073–1078. AAAI Press, 1994.

I. Little and S. Theibaux. Concurrent probabilistic planning in the graphplan framework. In Long and Smith [2006].

Iain Little, Douglas Aberdeen, and Sylvie Thiébaux. Prottle: A probabilistic temporal planner. In Veloso and Kambhampati [2005], pages 1181–1186.

M.L. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.

Michael L. Littman. Probabilistic propositional planning: Representations and complexity. In Ben Kuipers and Bonnie Webber, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island.*, pages 748–754. AAAI Press / The MIT Press, 1997.

Yaxin Liu, Sven Koenig, and David Furcy. Speeding up the calculation of heuristics for heuristic search-based planning. In Rina Dechter, Michael Kearns, and Rich Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, pages 484–491. AAAI Press, 2002.

Derek Long and Maria Fox. Efficient implementation of the plan graph in stan. *Journal of Artificial Intelligence Research*, 10:87–115, 1999.

Derek Long and Maria Fox. The 3rd International Planning Competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

Derek Long and Stephen Smith, editors. *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006), June 6-10 2006, The English Lake District, Cumbria, UK*. AAAI, 2006.

Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. In Jim Hendler and Devika Subramanian, editors, *Proceedings of the Sixteenth National Conference on Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, pages 541–548. AAAI Press / The MIT Press, 1999.

Stephen M. Majercik and Michael L. Littman. Maxplan: A new approach to probabilistic planning. In Reid G. Simmons, Manuela M. Veloso, and Stephen Smith, editors, *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, Pittsburgh Pennsylvania, USA.*, pages 86–93. AAAI, 1998.

S.M. Majercik and M.L. Littman. Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence*, 147(1-2):119–162, 2003.

Mausam and Daniel S. Weld. Concurrent probabilistic temporal planning. In Biundo et al. [2005], pages 120–129.

Mausam, Piergiorgio Bertoli, and Daniel S. Weld. A hybridized planner for stochastic domains. In Veloso [2007], pages 1972–1978.

Drew V. McDermott. A heuristic estimator for means-ends analysis in planning. In Drabble [1996], pages 142–149.

D.V. McDermott. PDDL-the planning domain definition language. Technical report, Available at: www.cs.yale.edu/homes/dvm, 1998.

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

Deborah L. McGuinness and George Ferguson, editors. *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*. AAAI Press / The MIT Press, 2004.

Nicloas Meuleau and David Smith. Optimal limited contingency planning. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 417–42, San Francisco, CA, 2003. Morgan Kaufmann.

A. Ricardo Morales, Phan Huy Tu, and Tran Cao Son. An extension to conformant planning using logic programming. In Veloso [2007], pages 1991–1996.

Jack Mostow and Charles Rich, editors. *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA*. AAAI Press / The MIT Press, 1998.

David J. Musliner, Edmund H. Durfee, and Kang G. Shin. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence*, 74(1):83–127, 1995.

B. Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.

XuanLong Nguyen and Subbarao Kambhampati. Extracting effective and admissible state space heuristics from the planning graph. In Henry Kautz and Bruce Porter, editors, *Proceedings of the*

Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA., pages 798–805. AAAI Press / The MIT Press, 2000.

XuanLong Nguyen, Subbarao Kambhampati, and Romeo Sanchez Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, 135(1-2):73–123, 2002.

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

N. Onder, G.C. Whelan, and L. Li. Engineering a conformant probabilistic planner. *Journal of Artificial Intelligence Research*, 25:1–15, 2006.

N. Onder. *Contingency Selection in Plan Generation*. PhD thesis, University of Pittsburgh, June 1999.

Héctor Palacios and Hector Geffner. Mapping conformant planning into sat through compilation and projection. In Roque Marín, Eva Onaindia, Alberto Bugarín, and José Santos, editors, *Current Topics in Artificial Intelligence, 11th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2005, Santiago de Compostela, Spain, November 16-18, 2005, Revised Selected Papers*, volume 4177 of *Lecture Notes in Computer Science*, pages 311–320. Springer, 2005.

Héctor Palacios and Hector Geffner. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*. AAAI Press, 2006.

Christos H. Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In Avrim Blum, editor, *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA.*, pages 86–92. IEEE Computer Society, 2000.

Edwin P. D. Pednault. ADL and the state-transition model of action. *Journal of Logic and Computation*, 4:467–512, 1994.

Mark Peot and David E. Smith. Conditional nonlinear planning. In James Hendler, editor, *Proceedings of the First International Conference on AI Planning Systems*, pages 189–197, College Park, Maryland, June 15–17 1992. Morgan Kaufmann.

Ronald P. A. Petrick and Fahiem Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In Ghallab et al. [2002], pages 212–222.

Joelle Pineau, Geoffrey J. Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for pomdps. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1025–1032. Morgan Kaufmann, 2003.

Ioannis Refanidis and Ioannis Vlahavas. The GRT planning system: Backward heuristic construction in forward state-space planning. *Journal of Artificial Intelligence Research*, 15:115–161, 2001.

Jussi Rintanen. Backward plan construction for planning with partial observability. In Ghallab et al. [2002], pages 173–183.

Jussi Rintanen. Expressive equivalence of formalisms for planning with sensing. In Giunchiglia et al. [2003], pages 185–194.

Jussi Rintanen. Distance estimates for planning in the discrete belief space. In McGuinness and Ferguson [2004], pages 525–530.

Jussi Rintanen. Conditional planning in the discrete belief space. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 1260–1265. Professional Book Center, 2005.

Wheeler Ruml, Minh Binh Do, and Markus P. J. Fromherz. On-line planning and scheduling for high-speed manufacturing. In Biundo et al. [2005], pages 30–39.

David E. Smith and Daniel S. Weld. Conformant graphplan. In Mostow and Rich [1998], pages 889–896.

F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0*. University of Colorado at Boulder, 1998.

Sam Steel and Rachid Alami, editors. *Recent Advances in AI Planning, 4th European Conference on Planning, ECP'97, Toulouse, France, September 24-26, 1997*, volume 1348 of *Lecture Notes in Computer Science*. Springer, 1997.

Felipe W. Trevizan, Fabio Gagliardi Cozman, and Leliane Nunes de Barros. Planning under risk and knightian uncertainty. In Veloso [2007], pages 2023–2028.

Le-Chi Tuan, Chitta Baral, Xin Zhang, and Tran Cao Son. Regression with respect to sensing actions and partial states. In McGuinness and Ferguson [2004], pages 556–561.

Menkes van den Briel, Romeo Sanchez Nigenda, Minh Binh Do, and Subbarao Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In McGuinness and Ferguson [2004], pages 562–569.

Manuela M. Veloso and Subbarao Kambhampati, editors. *Proceedings of The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*. AAAI Press / The MIT Press, 2005.

Manuela M. Veloso, editor. *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2007.

Thierry Vidal and Malik Ghallab. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In Wolfgang Wahlster, editor, *12th European Conference on Artificial Intelligence, Budapest, Hungary, August 11-16, 1996, Proceedings*, pages 48–54. John Wiley and Sons, Chichester, 1996.

Daniel S. Weld, Corin R. Anderson, and David E. Smith. Extending graphplan to handle uncertainty & sensing actions. In Mostow and Rich [1998], pages 897–904.

H.L.S. Younes and M.L. Littman. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical report, CMU-CS-04-167, Carnegie Mellon University, 2004.

H.L.S. Younes and R. Simmons. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405–430, 2003.

Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors. *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*. AAAI, 2004.

Terry Zimmerman and Subbarao Kambhampati. Using memory to transform search on the planning graph. *Journal of Artificial Intelligence Research*, 23:533–585, 2005.