A Tutorial on Planning Graph–Based Reachability Heuristics

Daniel Bryce and Subbarao Kambhampati

■ The primary revolution in automated planning in the last decade has been the very impressive scale-up in planner performance. A large part of the credit for this can be attributed squarely to the invention and deployment of powerful reachability heuristics. Most, if not all, modern reachability heuristics are based on a remarkably extensible data structure called the planning graph, which made its debut as a bit player in the success of GraphPlan, but quickly grew in prominence to occupy the center stage. Planning graphs are a cheap means to obtain informative look-ahead heuristics for search and have become ubiquitous in state-of-the-art heuristic search planners. We present the foundations of planning graph heuristics in classical planning and explain how their flexibility lets them adapt to more expressive scenarios that consider action costs, goal utility, numeric resources, time, and uncertainty.

Synthesizing plans capable of achieving an agent's goals has always been a central endeavor in AI. Considerable work has been done in the last 40 years on modeling a wide variety of plan-synthesis problems and developing multiple search regimes for driving the synthesis itself. Despite this progress, the ability to synthesize reasonable length plans under even the most stringent restrictions remained severely limited. This state of affairs has changed quite dramatically in the last decade, giving rise to planners that can routinely generate plans with hundreds of actions. This revolutionary shift in scalability can be attributed in large part to the use of sophisticated reachability heuristics to guide the planners' search.

Reachability heuristics aim to estimate the cost of a plan between the current search state and the goal state. While reachability analysis can be carried out in many different ways (Bonet and Geffner 1999, McDermott 1999, Ghallab and Laruelle 1994), one particular way—involving planning graphs—has proven to be very effective and extensible. Planning graphs were originally introduced as part of the GraphPlan algorithm (Blum and Furst 1995) but quickly grew in prominence once their connection to reachability analysis was recognized.

Planning graphs provide inexpensive but informative reachability heuristics by approximating the search tree rooted at a given state. They have also proven to be quite malleable in being adapted to a range of expressive planning problems. Planners using such heuristics have come to dominate the state of the art in plan synthesis. Indeed, 12 out of 20 teams in the 2004 International Planning Competition and 15 out of 22 teams in the 2006 International Planning Competition used planning graph–based heuristics. Lack of search control for domain-independent planners has, in the past, made most planning applications be written with a knowledge-intensive planning approach—such as hierarchical task networks. However, the effectiveness of reachability heuristics has started tilting the balance back and is giving rise to a set of applications based on domain-independent planners (Ruml, Do, and Fromherz 2005; Boddy et al. 2005).

This article[1] aims to provide an accessible introduction to reachability heuristics in planning. While we will briefly describe the planning algorithms to set the context for the heuristics, our aim is not to provide a compre-
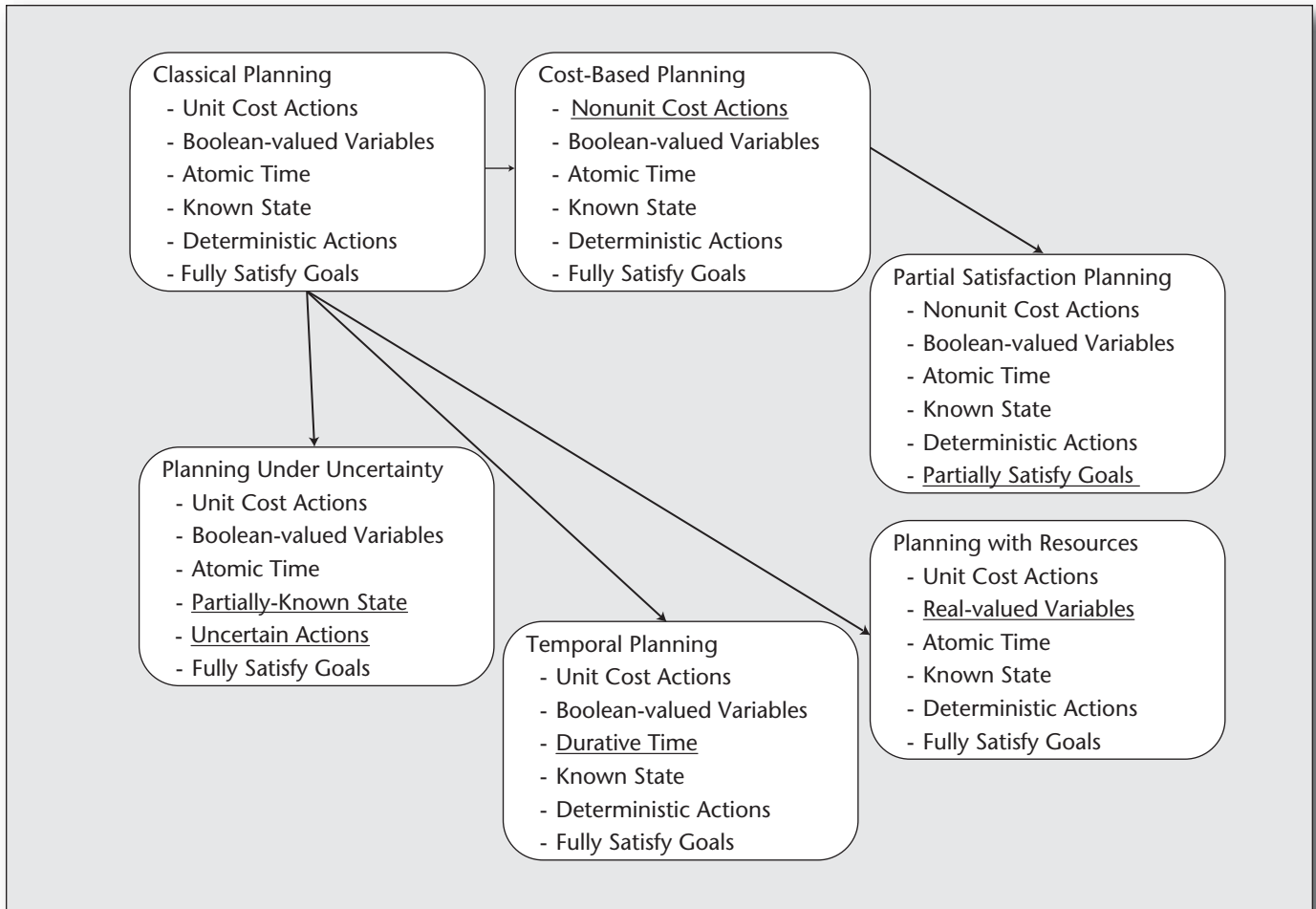
Figure 1. Taxonomy of Planning Models.

hensive introduction to planning algorithms. Rather, we seek to complement existing sources (see, for example, Ghallab, Nau, and Traverso [2004]) by providing a complete introduction to reachability heuristics in planning.

As outlined in figure 1, we discuss the classical planning model along with several extensions (in clockwise order). Naturally, as the planning model becomes more expressive, the planning graph representation and heuristics change. We start with classical planning to lay an intuitive foundation and then build up techniques for handling additional problem features. We concentrate on the following features independently, pointing out where they have been combined. Action costs are uniform in the classical model, but we later describe nonuniform costs. Goal satisfaction is total in the classical model but can be partial in general (that is, the cost of satisfying all goals can exceed the benefit). Resources are described only by Boolean variables in the classical model but can be integer or real-valued variables. Time is atomic in the classical model, but is

durative in general (that is, actions can have different durations). Uncertainty is nonexistent in the classical model but can generally result through uncertain action effects and partial (or no) observability. While there are many additional ways to extend the classical model, we hold some features constant throughout our discussion. The number of agents is restricted to only one (the planning agent), execution takes place after plan synthesis, and with the exception of temporal planning, plans are sequential (no actions execute concurrently).

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, with partial power reserves and some broken sensors. 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—each objective having a different utility. The rover must communicate any data it collects to have the objectives satisfied. The driving time and cost from one location to another vary depending on the terrain and distance between locations. Actions to collect soil, rock, and image data incur different costs and take various times. 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.

History of Reachability and Planning Graph 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 Drew 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 rediscovery of the idea of using reachability heuristics in planning was made by Blai Bonet and Hector Geffner (1999). Each rediscovery 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. Subbarao Kambhampati, Eric Lambrecht, and Eric Parker (1997) explicitly characterized the planning graph as an envelope approximation of the progression search tree (see figure 3 of his work and the associated discussion). Bonet and Geffner (1999) interpreted Graph-Plan as an IDA* search with the heuristic encoded in the planning graph. XuanLong Nguyen and Subbarao Kambhampati (2000) described methods for directly extracting heuristics from the 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.

Classical Planning

In this section we start with a brief background on how the classical planning problem is represented and why the problem is difficult. We follow with an introduction to planning graph heuristics for state-based progression search (extending plan prefixes). In the Heuristics in Alternative Planning Strategies section, we cover issues involved with using planning graph heuristics for state-based regression and plan space search.

Background

The classical planning problem is defined as a tuple $<P, A, S_I, G>$, where P is a set of propositions, A is a set of actions, S_I is an initial state, and G is a set of goal propositions. Throughout this article we will use many representational assumptions (described later) 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 initial state 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 a set of propositions pre(a) for execution preconditions, a set of propositions that it causes to become true eff$^+$(a), and a set of propositions it causes to become false eff$^-$(a). An action a is applicable $appl(a, s)$ to a state s if each precondition proposition holds in the state, pre(a) $\subseteq s$. The successor state s' is the result of executing an applicable action a in state s, where $s' = exec(a, s) = s \setminus$ eff$^-$ (a) $\cup$ eff$^+$(a). A sequence of actions $\{a_1, \ldots a_n\}$, executed in state s, results in a state s', where $s' = exec(a_n, exec(a_{n-1}, \ldots exec(a_1, s)\ldots))$ 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. For now we assume that actions have unit cost, making the cost of a plan equivalent to the number of actions.

We use the planning domain description

```
(define (domain rovers_classical)            (define (problem rovers_classical1)
 (:requirements :strips :typing)              (:domain rovers_classical)
 (:types location data)                       (:objects
 (:predicates                                   soil image rock - data
  (at ?x - location)                            alpha beta gamma - location)
  (avail ?d - data ?x - location)            (:init (at alpha)
  (comm ?d - data)                                  (avail soil alpha)
  (have ?d - data))                                 (avail rock beta)
 (:action drive                                     (avail image gamma))
  :parameters (?x ?y - location)             (:goal (and (comm soil)
  :precondition (at ?x)                                  (comm image)
  :effect (and (at ?y) (not (at ?x))))                   (comm rock)))
 (: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))
 )
```

Figure 2. PDDL Description of Classical Planning Formulation of the Rover Problem.

language (PDDL) (McDermott 1998) to describe STRIPS planning problems. Figure 2 is a PDDL formulation of the rover problem for classical planning.[2] 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 we use the objects defined in the problem description to instantiate predicates and action schemas. Grounding involves using every combination of objects to replace free variables in predicates to obtain propositions and in action schemas to obtain ground actions. The problem instance in figure 2 denotes that the initial state is:

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

and that the goal is:

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

The domain description in figure 2 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 {at(alpha)}, and it causes {at(beta)} to become true and {at(alpha)} to become false. Executing drive(alpha, beta) from the initial state results in the state:

$$s' = exec(drive(alpha, beta), s_I) \\ = \{at(beta), avail(soil, alpha), \\ avail(rock, beta), avail(image, gamma)\},$$

because at(beta) becomes true and at(alpha) becomes false. A valid plan for the problem in figure 2 is 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 for a shortest path). 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.[3] 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, in the worst case, it is still possible to construct the entire transition graph. Heuristic search algorithms, such as A* search, can "intelligently" search for plans and, we hope, 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 (also known as 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 and 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 our example is depicted in figure 3a. The projection is represented as states in dark boxes connected through edges for actions. The propositions holding in each state are list-ed (except for the avail propositions, which are in all states).[4] Within this tree, the exact reachability cost for each 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, we have to explore 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 our purpose quite well. Figure 3b 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 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 effect have(soil). 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 that 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 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, planning graph data structure is pro-
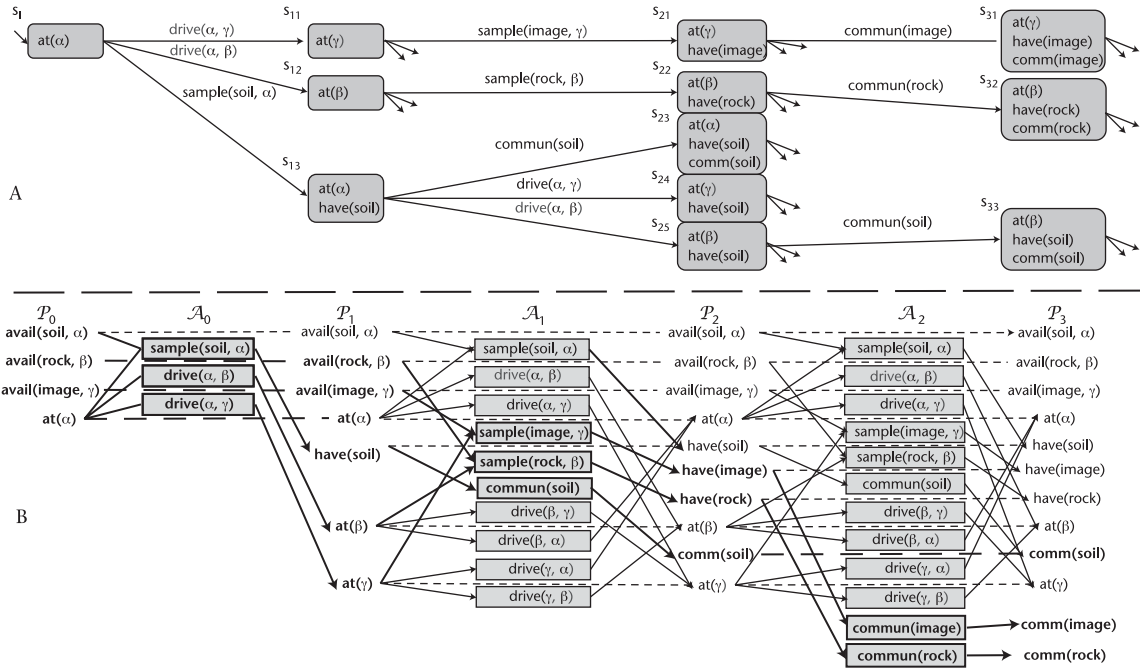
Figure 3. Progression Search Graph (a) and Planning Graph (b) for the Rover Problem.

viding optimistic reachability estimates.

We can view the planning graph 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 (that is, 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 as 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) through 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

We 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 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, we simplify our 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[5] of an action in $\mathcal{A}_{i-1}$. It is common to use implicit actions for proposition persistence (also known as 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 $\mathrm{pre}(a_p) = \mathrm{eff}^+(a_p) = p$.

Planning graph construction can continue until one of the following conditions holds: (1) the graph has *leveled off* (that is, two subsequent proposition layers are identical), or (2) the goal is reachable (that is, every goal proposition is present in a proposition layer).

In figure 3a 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 for a transition between two states, a source state 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 plans (Nguyen, Kambhampati, and Nigenda 2002). The most obvious level-based heuristic, called the set-level heuristic, estimates the plan cost to reach a goal state by finding the first level where the proposition layer includes all of the propositions in the goal. This level index is used as the heuristic. Other 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, through maximization ($\max_{g \in G} c(s, g)$) or summation ($\Sigma_{g \in G} c(s, g)$). The heuristics are level based because the cost 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). We explore both types of heuristics in detail.

To illustrate the different heuristics, we will use three goals: G = {comm(soil), comm(image), comm(rock)} (the original goal in the example), 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 we assess the cost of the set of propositions, we make additional assumptions.

The set-level heuristic is defined as the index of the first proposition layer where all goal propositions appear. For example, the cost of G_1 is 2, because both propositions first appear together in $\mathcal{P}_2$. The set-level heuristic assumes that the subplans to individually achieve the goal propositions *positively interact*, which is appropriate because the rover must be at(beta) in order to have(rock). Positive interaction captures the notion that actions taken to achieve one goal will simultaneously help achieve another goal. Positive interaction is not always the right assumption: the cost to reach G_2 is 1 because both propositions appear in $\mathcal{P}_1$ but they require different actions for support. The set-level heuristic for the cost to reach G is 3 because the goal propositions are not reachable together until $\mathcal{P}_3$ of $PG(s_I)$. The set-level heuristic never overestimates the cost to achieve a set of propositions and is thus admissible. As we will see in the next section, we can adjust the set-level heuristic with mutexes to incorporate negative interactions and strengthen the heuristic.

As an alternative to the set-level heuristic, cost can be assigned to each individual proposition in the goal. Aggregating the cost for each proposition provides a heuristic. Using a max-

Heuristic	G	G_1	G_2
Set-Level	3	2	1
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.*

The goals are G = {comm(soil), comm(image), comm(rock)}, G_1 = {at(beta),
have(rock)}, and G_2 = {at(beta), have(soil)}.

imization assumes that the actions used to achieve the set of propositions will positively interact, like the set-level heuristic. For example, with G_1 it is possible to achieve at(beta) in $\mathcal{P}_1$ defining $c(s_I, \text{at(beta)}) = 1$, and achieve have (rock) in $\mathcal{P}_2$, defining $c(s_I, \text{have(rock)}) = 2$. 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 original goal G from s_I is 3, because $\max(c(s_I, \text{have(soil)}), c(s_I, \text{have(rock)}), c(s_I, \text{have(image)})) = \max(2,3,3) = 3$. At this point, the set-level and max heuristics are identical. We will see in the next section how set-level more readily incorporates negative interactions between goal propositions to improve the heuristic (making it different from the max heuristic). Conversely, in the section on cost-based planning, we will see how the max heuristic and sum heuristic (described next) generalize easily because they are already defined in terms of proposition costs.

Using a summation to aggregate proposition costs assumes that the subplans 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 $c(s_I, \text{at(beta)}) = 1$, and achieve have(soil) in $\mathcal{P}_1$, defining $c(s_I, \text{have(soil)}) = 1$. 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 $c(s_I, \text{have(soil)}) + c(s_I, \text{have(rock)}) + c(s_I, \text{have(image)}) = 2 + 3 + 3 = 8$. In practice the sum heuristic usually outperforms the max heuristic, but gives up admissibility.

The primary 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 (described next), by using cost propagation (described in the Cost-Based Planning section), or by using a serial planning graph (described in the next section).

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 the number of actions in a plan. Through a simple back-chaining algorithm (figure 4) 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, we 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: (1) 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 (2) every action $a \in \mathcal{A}_i^{RP}$ in the relaxed plan has its preconditions pre$(a) \subseteq \mathcal{P}_i^{RP})$ 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 A_0^{RP} because they both help support the goal, despite conflicting.

Figure 4 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. Choosing actions in 7 is the most critical step in the algorithm. We will see in later sections, such as the section on cost-based planning, that it is possible to supplement the planning graph with additional information that can bias the action choice in line 7. Lines 10–12 insert the preconditions of chosen actions into the relaxed plan. The algorithm ends by returning the relaxed plan. The relaxed plan heuristic 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 3b 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, so the heuristic is 8. In line 7 of figure 4, 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.

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

$(\mathcal{A}_0^{RP}$ = {drive(alpha, beta)},

$\mathcal{A}_1^{RP}$ = {sample(rock, beta)}), and

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

In these, 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 one or the other).

Relaxed plan extraction can be quite fast, mostly due to the fact that the extraction can be done by a *backtrack free* choice of actions at each level. It is possible to find the optimal relaxed plan by using line 7 in figure 4 as a backtrack point in a branch and bound scheme. In general, finding an optimal relaxed plan is NP-hard. Recent work (Do, Benton, and Kambhampati 2006) has noticed that, like the original GraphPlan search for plans, finding relaxed plans can be posed as combinatorial search, solvable by optimizing or satisficing algorithms. While the procedural approach (described above) is practical in classical planning, using combinatorial search may better

```
RPExtract(PG(s), G)
 1: Let n be the index of the last level of PG(s)
 2: for all  p ∈ G ∩ 𝒫_n do /* Initialize Goals */
 3:     𝒫_n^RP ← 𝒫_n^RP ∪ p
 4: end for
 5: for i = n ... 1 do
 6:     for all p ∈ 𝒫_i^RP do /* Find Supporting Actions */
 7:         Find  a ∈ 𝒜_{i-1} such that p ∈ eff⁺(a)
 8:         𝒜_{i-1}^RP ← 𝒜_{i-1}^RP ∪ a
 9:     end for
10:     for all  a ∈ 𝒜_{i-1}^RP, p ∈ pre(a) do /* Insert Preconditions */
11:         𝒫_{i-1}^RP ← 𝒫_{i-1}^RP ∪ p
12:     end for
13: end for
14: return  (𝒫_0^RP, 𝒜_0^RP, 𝒫_1^RP, ..., 𝒜_{n-1}^RP, 𝒫_n^RP)
```

Figure 4. Relaxed Plan Extraction Algorithm.

incorporate cost, utility, and other factors described in later sections. Nevertheless, we will see that procedural relaxed plan extraction can be biased in several ways to handle additional problem constraints, such as action costs. As with all heuristics, the quality and computation time play competing roles, leaving the dominance of procedural versus combinatorial algorithms, as yet, unsettled. Regardless of the extraction algorithm, relaxed plans tend to be very effective in practice and form the basis for most modern heuristic search planners.

Related Work

Level-based heuristics were first introduced by Nguyen and Kambhampati (2000, 2001), while relaxed plan heuristics became popular with the success of the FF planner (Hoffmann and Nebel 2001). There are a number of improvements and alternatives that can be made beyond the basic ideas on planning graph construction and heuristic extraction. One common extension is to limit the branching factor of search using actions occurring in the planning graph or relaxed plan. The idea is to restrict attention to only those actions whose preconditions can be satisfied in a reachable state. A conservative approach would allow search to use only actions appearing in the last action layer of a leveled-off planning graph (a complete strategy). The actions that do not appear in these layers will not have their preconditions reachable and are useless. A less

conservative approach uses only those actions in the layer preceding the layer where the goals are supported (an incomplete strategy) (Nguyen, Kambhampati, and Nigenda 2002). An even more aggressive and incomplete strategy called "helpful actions" (Hoffmann and Nebel 2001) involves applying only those actions that have effects similar to actions chosen for the first step of the state's relaxed plan. In the rover example, search with helpful actions would only apply actions to s_I that have at least one of the propositions {have(soil), at(beta), at(gamma)} in their effect because these propositions appear in $\mathcal{P}_1^{RP}$.

Another use for relaxed plans is to derive macro actions (plan fragments). The YAHSP planner (Vidal 2004) encodes relaxed plans into a CSP to resolve action conflicts, creating a macro action. This often leads to search making very few (albeit expensive) choices.

While the planning graph was originally introduced as part of the GraphPlan search algorithm and subsequently used for heuristics in state-based search, GraphPlan-based search has been revived in the search employed by LPG (Gerevini, Saetti, and Serina 2003). LPG uses local search on the planning graph to transform a relaxed plan into a feasible plan. LPG uses relaxed plan heuristics to evaluate potential plan repairs that add and remove actions.

Adjusting for Negative Interactions

We noted that the proposition layers in the planning graph are an upper bound approximation to the states in the projection tree. Specifically, while every legal state is a subset of the proposition layer, not every subset of the proposition layer corresponds to a legal state. Because of the latter, the reachability estimates can be too optimistic in scenarios where there are negative interactions between the actions. There is a natural way to tighten the approximation—mark subsets of the proposition layers that cannot be present together in any legal state. Any subset of the proposition layer that subsumes one of these "mutual exclusion" sets will not correspond to a legal state (and thus should not be considered reachable by that level). The more mutual exclusion sets we can mark, the more exact we can make the planning graph–based reachability analysis. Intuitively, if all mutual exclusion sets of all arity are marked, then the planning graph proposition layers correspond exactly to the projection tree states. The important point is that the marking procedure does not need to be all or

nothing. As long as the markup is "sound" (in that any subset marked mutually exclusive is indeed mutually exclusive), the planning graph continues to provide an optimistic (lower-bound) estimate on reachability.

The question is whether it is possible to mark such mutual exclusions efficiently. It turns out that it is indeed feasible to compute a large subset of the binary mutual exclusions (henceforth called "mutexes") in quadratic time using a propagation procedure. Mutexes arise in action layers, where they record which of the actions cannot be executed in parallel (because they disagree on whether a proposition should be positive or negative in an effect or precondition). Mutex propositions result from mutexes among supporting actions when there is no set of nonmutex actions supporting the propositions.[6]

To illustrate, figure 5 shows mutexes for the first two levels of the planning graph $PG(s_I)$. Mutexes are denoted by the arcs between actions and between propositions. Notice that the two drive actions in $\mathcal{A}_0$ are mutex with all other actions and the persistence of at(alpha). Both drive actions make at(alpha) false, interfering with each action requiring that at(alpha) be true. This results in proposition mutexes between have(soil) and both at(beta) and at(gamma) in $\mathcal{P}_1$. Mutexes capture the fact that the rover cannot be at(beta) and have(soil) after one step. However, notice that it is possible after two steps for the rover to be at(beta) and have(soil). Mutexes effectively reduce the optimism about which states are reachable, tightening the approximation of the progression search graph.

A stronger interpretation and computation of mutex propagation takes place in the previously mentioned serial planning graphs, which remove all notion of action parallelism by forcing each action to be mutex with every other action (except for noop actions). The serial planning graph has the advantage of improving level-based heuristics by more tightly correlating level and cost. However, serial planning graphs are not often used in practice because they require a large number of additional mutexes (and also make the planning graph achieve level-off much later). In the remainder of this section, we describe the "normal" planning graph with mutexes, where action parallelism is allowed.

Heuristic Adjustments

It is possible to improve the quality of planning graph heuristics by adjusting them to make up for ignoring negative interactions. We started with a relaxation of the projection tree that
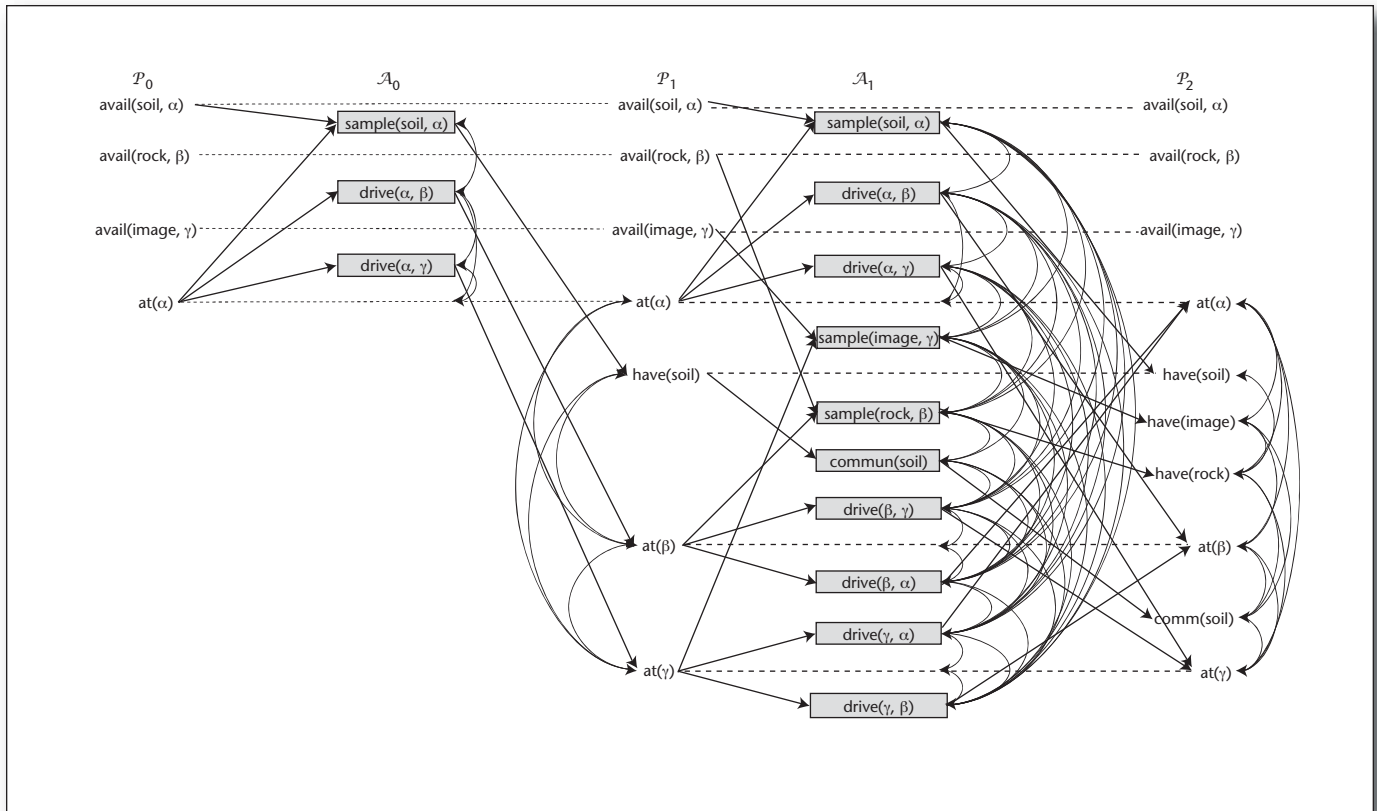
Figure 5. Classical Planning Graph with Binary Mutexes.

ignored all negative interactions, in favor of causal support, to estimate the achievement cost of subgoals. While this relaxation works well for many problems and search strategies, there are ways to either *strengthen* or adjust the heuristic relaxation to better guide search. For instance, it is possible to strengthen the set-level heuristic because it is defined as the index of the first level where all goal propositions appear reachable. If any pair of the goal propositions is mutex in a level, then even if all goal propositions are present the goal is not reachable. Set level is still admissible with mutexes, and becomes a better lower bound on the cost to reach the goal. What is perhaps more interesting is that admissibility depends only on the soundness but not the completeness of mutex propagation. We can thus be lazy in propagating mutexes. For example, consider the set-level heuristic for $G_2 = \{at(beta), have(soil)\}$ with and without mutexes. With mutexes the heuristic value is 2 because the propositions do not appear together without a mutex until $\mathcal{P}_2$, whereas without using mutexes the heuristic value is 1. This illustrates the difference between the max and set-level heuristics because without mutexes the max heuristic is equal to the set-level heuristic and with mutexes they are different.

Alternatively we can *adjust* the relaxed plan heuristic, to get the adjusted sum heuristic, by adding a penalty factor to account for ignoring negative interactions. One penalty factor can be the difference between the set-level heuristic and the max heuristic: indicating how much extra work is needed to achieve the goals *together*. While the adjusted sum heuristic is costly because it involves computing a relaxed plan, set level, and max heuristic, it guides search very well and improves performance on large problems (especially in regression search, described in the next section).

The notion of adjusting heuristics to account for ignored constraints is called phased relaxation. As we will see in later sections, including more problem features makes it more difficult to include everything in planning graph construction. Phased relaxation allows us to get a very relaxed heuristic and make it stronger.

Related Work

Mutexes were originally introduced in Graph-Plan (Blum and Furst 1995), but Nguyen and Kambhampati (2000) realized their application to adjusting heuristics. Helmert (2004) describes another way to include negative interactions in reachability heuristics. Planning domains best expressed with multivalued
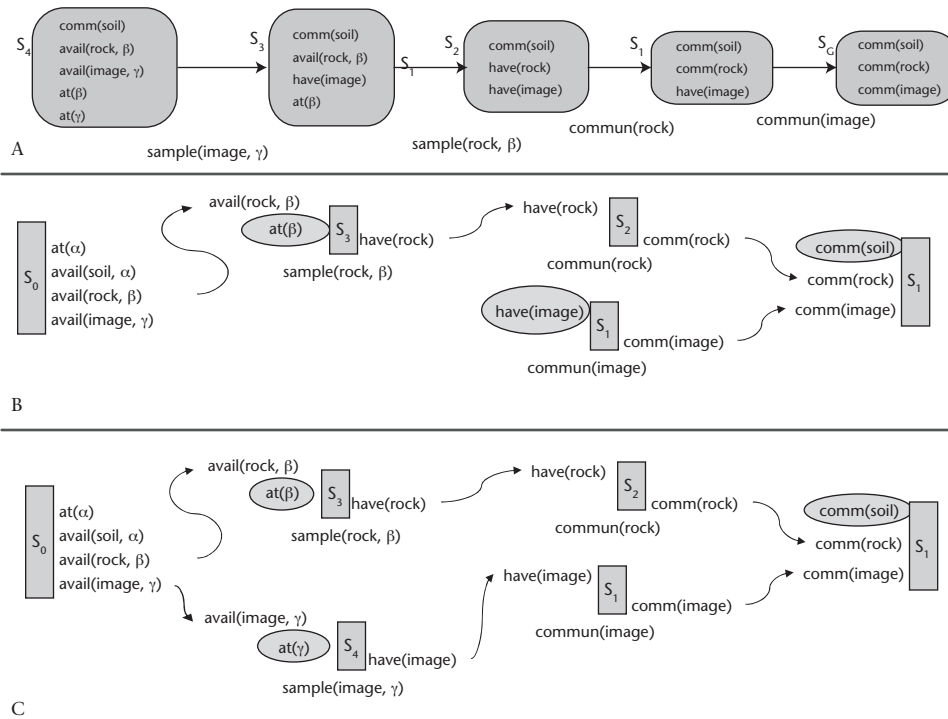
Figure 6. State Space and Plan Space Regression.

variables (as in the SAS+ language [Backstrom and Nebel 1995]) are often described in the classical planning formalism with Boolean variables (such as the at(?x) predicate in the example). By representing an inherently multivalued variable as Boolean, information about negative interactions is lost (for example, that at(alpha) and at(beta) are impossible together), where it is readily available in the multivalued encoding (for example, at = alpha or at = beta because at can have only one value). By translating a Boolean encoding of a problem to a multivalued encoding, it is possible to solve different relaxed planning tasks (which incorporate these negative interactions). As yet, there is no decisive result indicating whether Boolean or multivalued variable encodings are best, much as there still exists the same question with respect to SAT and CSP.

Another recent work (Yoon, Fern, and Givan 2006) automatically learns adjustments for relaxed plan heuristics. The idea is to learn a linear regression function from training plans that reflect the difference in relaxed plan length and the actual cost to reach the goal.

Heuristics in Alternative Planning Strategies

Planning graph heuristics can be used for many different types of search, including regression and plan space search. Regression and plan space search are similar, in that both use means-ends reasoning to find actions that are relevant to supporting the goals. We have presented planning graph heuristics in terms of progression search, where we estimate the cost of a plan to transition between two sets of propositions (a search state and the goal). We can use the same technique for means-ends reasoning by carefully redefining the two sets of propositions. The initial state replaces the search state, and regression states replace the goal.

Regression Search

Regression search (or back chaining) constructs plan suffixes starting from the goal. Because the planning problem defines a goal by a set of propositions instead of a state, there may be

several goal states. Rather than individually search for a plan for each goal state, it is common to search for all goal states at once in the space of partial states. Each partial state is characterized by a set of propositions that are known to be true and a set of propositions that are known to be false, assuming all other propositions are unknown. Since each proposition is true, false, or unknown, there are $3^{|P|}$ partial states. Regression in the space of partial states identifies plan suffixes that will reach *a goal state*. The regression tree is rooted at the partial goal state, and leaf nodes are partial states. The path from a leaf node to the root is a plan suffix that guarantees any state consistent with the leaf node can reach a goal state by executing the plan suffix. A valid plan starts with a partial state that is satisfied by the initial state.

Figure 6 illustrates a regression search path (figure 6a) and two partial plan space plans (figures 6b and 6c), described later in the section, for the rover example. The regression search trace starts on the right with a partial state comprised of the goal propositions. Each partial state is generated by reasoning about what must be true in the previous state for the applied action to make the latter state exist. For example, s_1 is the partial state resulting from regressing the partial goal state s_G with commun(image). Because commun(image) does not make any proposition in s_G false and it makes comm(image) true, it is applicable to s_G. To construct s_1, regression removes the positive effect comm(image) and adds the precondition have(image).

Heuristics for Regression

Planning graphs can estimate the cost of a plan to transition between a complete state and a partial state. In progression, search encounters several complete states (each requiring a planning graph), and each must reach a partial goal state. In regression, search generates several partial states, and each must be reached from the complete initial state (requiring a single planning graph). Planning graphs are single-source, multiple-destination relaxations of the projection tree. In progression, the source is constantly changing, depending on the search state, and the destination is always the goal. In regression, the source is constant, but the destination changes, depending on the regressed partial state. While it may seem to the reader that regression has a definite advantage because it needs only one planning graph, we will shortly see that the situation is more complex.

The planning graph in figure 3b can be used to compute the heuristic for any regressed partial state in the example. To compute a level-based heuristic for s_3, we find comm(soil) first appears in $\mathcal{P}_2$, avail(rock, beta) appears in $\mathcal{P}_0$, have(image) appears in P_2, and at(beta) appears in $\mathcal{P}_1$. Using the level-based sum heuristic to measure its cost provides $2 + 0 + 2 + 1 = 5$. Similarly, for state s_4, the same heuristic provides $2 + 0 + 0 + 1 + 1 = 4$.

The reader should notice that the s_4 regression state in figure 6 is inconsistent because it asserts that the rover is at two locations at once. Regression search allows inconsistent / unreachable states. The heuristic measure of this inconsistent state is 4, but it should be ∞ because it cannot be reached from the initial state. The search may continue fruitlessly and explore through this state unaware that it will never become a valid solution. One can correct this problem by propagating mutexes on the planning graph and using the set-level heuristic. With set level there is no level where at(beta) and at(gamma) are not mutex, so the heuristic value is ∞. Nguyen and Kambhampati (2001) show that the adjusted sum heuristic discussed earlier—which adds a penalty based on negative interactions to the relaxed plan heuristic—does quite well in regression search. However, this analysis increases the computational cost of planning graph construction.

Progression Versus Regression

Table 2 summarizes the many important differences between progression and regression from the point of view of reachability heuristics. In progression, the search encounters several complete states (at most $2^{|P|}$), and one of them must reach a partial goal state. Regression search encounters several partial states (at most $3^{|P|}$), and one of them must be reached from the complete initial state. Progression requires a planning graph for each encountered state, whereas regression requires a single planning graph for the initial state of the problem. Mutexes play a larger role in regression by improving the ability of search to prune inconsistent partial states. However, binary mutex analysis is often not enough to rule out all inconsistent states, and considerably costlier higher-order mutex propagation may be required. Progression states are always consistent and mutexes only help adjust the heuristic. Thus, there is a trade-off between searching in the forward direction constructing several inexpensive planning graphs and searching in the backward direction with a single costly planning graph. Current planner performance indicates that progression works better.

Feature	Progression	Regression
Search Nodes	Complete/Legal States	Incomplete/Possibly Inconsistent
Search Space Size	$2^{|P|}$	$3^{|P|}$
Number of Planning Graphs	$O(2|P|)$	1
Mutexes	Marginally Helpful	Extremely Helpful

Table 2. Comparison of Planning Graphs for Progression and Regression Search.

Plan Space Search

Plan space search (also known as partial-order or causal-link planning) shares many similarities with regression search and it too can benefit from planning graph heuristics. Both plan space and regression search reason backward from the goal to the initial state, but plan space search does not maintain an explicit notion of state. Rather, plan space search algorithms reason about which actions causally support the conditions of other actions or the goals.

Any partial plan space plan can be characterized in terms of its open conditions: those preconditions or goals that do not yet have causal support. The open conditions may never all need to hold in a single state at the same time, but we can think of them as an approximate regression state. In particular, by assuming that negative interactions do not exist, it is always possible to achieve all conditions in the beginning of the plan and persist them to where they are needed. With this approximate state, we can measure its reachability cost from the initial state just as in regression search (Nguyen and Kambhampati 2001; Younes and Simmons 2003).

In figure 6b and 6c, each plan space plan is represented by a set of steps, causal links, and ordering relations. Each step has an associated action with preconditions and effects. We denote each step by a rectangle, with the preconditions on the left and effects on the right. The open conditions are contained in ovals. The regression state s_3 contains every proposition that is an open condition in the plan space plan in figure 6b, and likewise for the s_4 and plan space plan in figure 6c. The heuristic values for the plan space plans in figure 6 are the same as the analogous regression states because their open conditions are identical to the propositions that hold in the regression states.

Related Work

Planning graph heuristics in regression search were first considered by Nguyen and Kambhampati (2000, 2001). Their application to partial-order planning was first considered by Nguyen and Kambhampati (2001) and subsequently by Younes and Simmons (2003).

GraphPlan itself also benefits from planning graph heuristics. By adopting the view that GraphPlan solves a constraint satisfaction problem where variables are propositions and values are supporting actions, planning graph heuristics help define variable and value ordering heuristics. The GraphPlan value ordering heuristic is to consider only those supporters of a proposition that appear in the previous action layer. Many works have improved upon the GraphPlan search algorithm by using heuristics similar to those just described. For instance, Kambhampati and Sanchez (2000) recognized how level-based heuristics can be used in the ordering heuristics in GraphPlan. Defining proposition cost as before, it is preferable to support the most costly propositions first to extract plans starting at later levels. Extracting plans at earlier levels first can lead to costly backtracking. Defining action cost as the aggregate cost of its precondition propositions, it is preferable to support a proposition with the least costly actions first. As we will see in the next section, these heuristics are related to our intuitions for propagating cost in the planning graph. The PEGG planner (Zimmerman and Kambhampati 2005) uses similar planning graph heuristics in its search. PEGG links GraphPlan search with state space search by identifying the states consistent with the steps in a partial GraphPlan solution. Using the identified states, PEGG can get better heuristics to rank partial solutions and guide GraphPlan search.

Some less-traditional GraphPlan algorithms

also use reachability heuristics. Least Commitment Graph-Plan (LCGP) (Cayrol, Regnier, and Vidal 2000) uses heuristics similar to Kambhampati and Sanchez (2000) in a GraphPlan algorithm that alters the structure of the planning graph. LCGP reduces the number of levels in the planning graph by allowing more parallelism; mutex actions can appear in the same level if there is a feasible serialization. The advantage of LCGP is in using fewer levels to find the same length plan, thus reducing the cost of constructing the planning graph. The work of Hoffmann and Geffner (2003) exploits the connection between GraphPlan and constraint satisfaction problems to find alternative branching schemes in the BBG planner. BBG allows action selection at arbitrary planning graph layers until a consistent plan is found. BBG guides its search with a novel level-based heuristic. Each partial plan is a set of action choices in different planning graph layers, so the heuristic for each partial plan constructs a planning graph respecting the partial plan action choices. For instance, BBG would exclude all actions that are mutex with a chosen action from the planning graph, favoring its commitments. The number of levels in the resulting planning graph is the heuristic merit of the partial plan.

We mentioned that a key difference between progression and regression is the number of planning graphs required. The backward planning graph (Refanidis and Vlahavas 2001) allows progression search to use a single planning graph, similar to regression. The backward planning graph reverses the planning graph by identifying states reachable by regression actions from the goal. In practice, the reverse planning graph suffers from the difference between the initial state and goal descriptions: the former is complete (that is, consistent with a single state), while the latter is partial (that is, consistent with many states). Because there are multiple goal states, reachability analysis is weakened, having fewer propositions to which actions are relevant.

Planning graph heuristics can be computed in alternative ways that were originally discovered in the context of partial-order planning and regression. IxTeT (Ghallab and Laruelle 1994) and UNPOP (McDermott 1996, 1999) compute the reachability heuristics on demand using a top-down procedure that does not require a planning graph. Given a state *s* whose cost to reach the goal needs to be estimated, both IxTeT and UNPOP perform an approximate version of regression search (called "greedy regression graphs" in UNPOP) to estimate cost of that state. The approximation involves making an independence assumption and computing the cost of a state s as the sum of the costs of the individual literals comprising *s*. When a proposition *p* is regressed using an action *a* to get a set of new subgoals *s′*, *s′* is again split into its constituent propositions. UNPOP also has the ability to estimate the cost of partially instantiated states (that is, states whose predicates contains variables).

HSP (Bonet and Geffner 2000a) and HSP-r (Bonet and Geffner 1999) use a bottom-up approach for computing the cost measures, again without the use of a planning graph. They use an iterative fixed-point computation to estimate the cost of reaching every literal from the given initial state. The computation starts by setting the cost of the propositions appearing in the initial state to 0, and the cost of the rest of the propositions to ∞. The costs are then updated using action application until a fixed point is reached. The updating scheme uses independence assumptions in the following way: if there is an action *a* that gives a proposition *p*, and *a* requires the preconditions $p_1, p_2, ..., p_i$, then the cost of achieving *p* is updated to be the minimum of its current cost, and the sum of the cost of *a* and the sum of the costs of achieving $p_1, p_2, ..., p_i$.

Once this updating is done to the fixed point, estimating the cost of a state *s* involves summing the costs of the propositions composing *s*. In later work, Haslum and Geffner (2000) point out that this bottom-up fixed-point computation can be seen as an approximate dynamic programming procedure, in which the cost of a state *s* is estimated as the maximum of the costs of its *k*-sized subsets of propositions.

These approaches can be related to planning graph heuristics. Planning graph heuristics can be seen as a bottom-up computation of reachability information. As discussed in Nguyen, Kambhampati, and Nigenda (2002, section 6.3), basing the bottom-up computation on the planning graph rather than on an explicit dynamic programming computation can offer several advantages, including the access to relaxed plans and help in selecting actions.

Cost-Based Planning

The rover problem is coarsely modeled in the classical planning framework because there are many seemingly similar quality plans that are actually quite different. The first limitation we lift is the assumption that actions have unit costs. In the rover example with unit cost actions, there are two equivalent plans to achieve the goal *G*. The first visits beta then
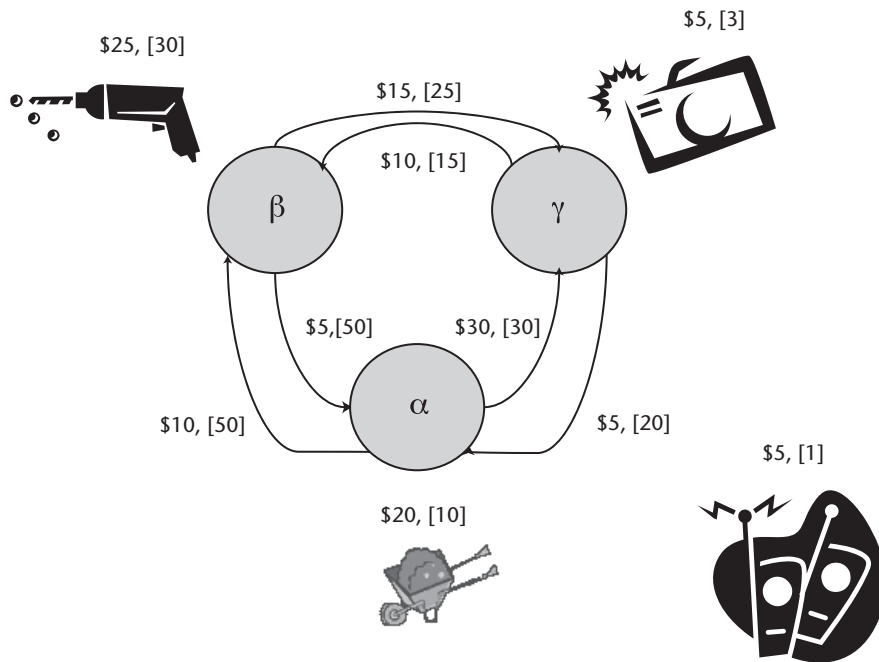
Figure 7. Costs ($) and Durations ([]) for Actions.

gamma, and the second visits gamma then beta. However, with the cost model illustrated in figure 7, there is a difference between these two plans. Figure 7 graphically depicts the cost (used here) and duration (used later) of each action. Driving to gamma then beta is 15 units costlier than driving to beta then gamma.

The extension to planning graph heuristics to consider nonuniform action costs is to propagate cost functions in the planning graph. Where previously cost was measured using the planning graph–level index, cost is now less correlated with the level index, thus requiring cost functions. Reachability heuristics measure the cost of achieving a set of propositions with the cost functions. This requires explicitly tracking the cost of achieving a proposition at each layer. Shortly, we will describe how heuristics change, but we consider first how one can propagate cost functions in the planning graph.

Cost Functions

Cost functions capture the distinction between reachability cost and proposition-layer indices

by propagating the best cost to achieve each proposition at each level. Figure 8 illustrates the planning graph $PG(s_I)$ with cost functions. The cost above each proposition indicates its achievement cost, and the cost above each action is the cost to support and execute the action. Notice that it takes only one level to achieve at(gamma), but it requires the drive(alpha, gamma) action whose cost is 30. The cost functions allow a possibly different cost for a proposition at different levels. For instance, at(gamma) has cost 30 in $\mathcal{P}_1$ and cost 25 in $\mathcal{P}_1$ because it can be achieved with a different, lower cost in two levels. With unit-cost actions, achieving a proposition in fewer levels is usually best, but with non-unit-cost actions, a least cost plan may use more levels.

Cost functions are computed during planning graph expansion with a few simple propagation rules. Propositions in the initial proposition layer have a cost of zero. Each action in a level is assigned the sum of its execution cost and its support cost. An action's support cost reflects the cost to reach its set of precondition propositions. It is typical to define the cost of a

comm(soil)	comm(rock)	comm(image)	Cost	Utility	Net Benefit
			0	0	0
		√	35	20	-15
	√		40	60	20
	√	√	65	80	15
√			25	50	25
√		√	60	70	10
√	√		**65**	**110**	**45**
√	√	√	90	130	40

Table 3. The Net Benefit for Each Goal Subset with Respect to an Optimal Cost Plan for the Subset.

The optimal net benefit is shown in bold.

set of propositions with a maximization (used in max-propagation) or summation (used in sum-propagation) of the individual proposition costs (similar to the max and sum heuristics). Finally, the cost to reach a proposition is the same as its least cost supporting action from the previous level.

Figure 8 depicts the planning graph with cost functions propagated from the initial state, using max-propagation. The goals comm(soil) and comm(rock) have minimum cost in the proposition layer where they first appear ($\mathcal{P}_2$ and $\mathcal{P}_3$, respectively), but comm(image) does not have minimum cost until one extra layer, $\mathcal{P}_4$, after it first appears in $\mathcal{P}_3$. The comm(image) goal proposition becomes less costly after another level because it is more costly to drive directly to gamma from alpha (one step), than it is to drive from alpha to beta and beta to gamma (two steps).

Notice that costs monotonically decrease over time. Since cost propagation always chooses the cheapest supporter for a proposition and the set of supporters monotonically increases, it is only possible for cost to decrease or remain constant. The cost of every proposition in figure 8 decreases, and eventually remains constant, leading us to several strategies for ending cost propagation.

Ending Cost Propagation

The original termination criterion for planning graph construction is no longer sufficient because the cost of each proposition may decrease at each level. Stopping when the goal propositions first appear may overestimate plan cost when they have lower costs at later levels. Stopping when two proposition layers are identical (the graph is leveled off) does not prevent an action's cost from dropping in the last action layer, making a goal proposition have lower cost. To allow costs to stabilize, we can generalize the leveling-off condition to use ∞-lookahead by extending the planning graph until proposition costs do no change. In some cases, using ∞-lookahead may be costly because costs do not stabilize for several levels. An approximation to ∞-lookahead is to use k-lookahead by extending the planning graph an additional k levels after the goal propositions first appear. The example, depicted in figure 8 uses 1-lookahead (extending the planning graph one level past the first appearance of all goal propositions).

Cost-based Heuristics

Level-based heuristics no longer accurately model the cost to reach the goal propositions because actions are no longer unit cost. Cost-based heuristics generalize level-based heuristics by using cost functions instead of level indices to define proposition reachability costs. With cost functions the heuristics care about the minimum cost to achieve a proposition instead of its first level. The minimum cost is always the cost at the last level of the planning graph because costs decrease monotonically. In the example depicted in figure 8, at(gamma) has cost 30 after one step because it can only be supported by drive(alpha, gamma), but after
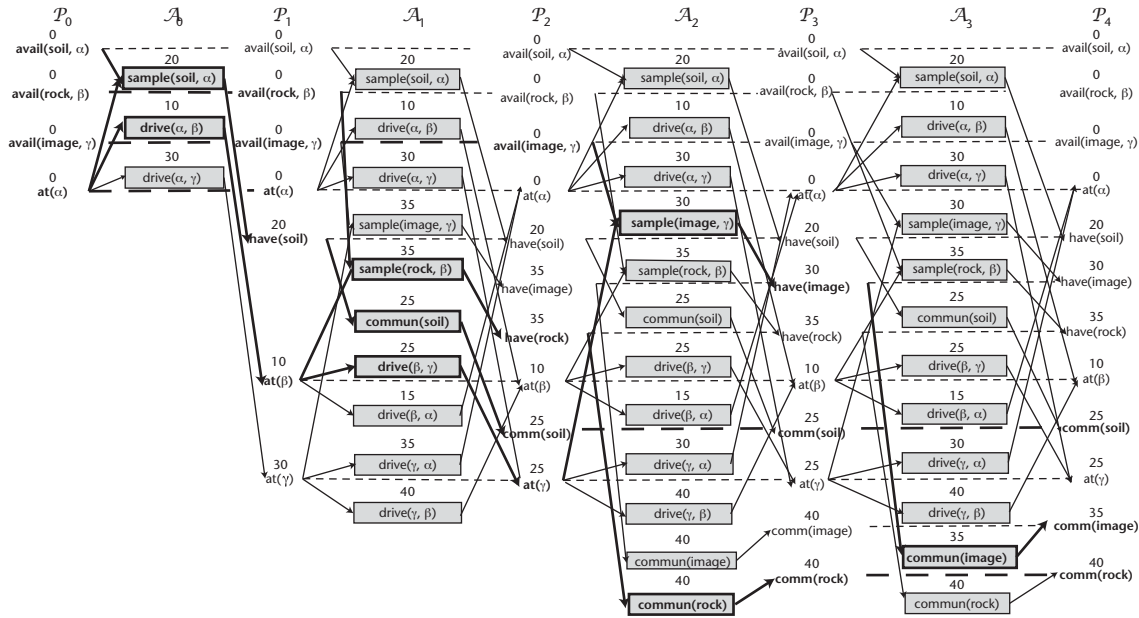
Figure 8. Cost Propagation for the Portion of the Planning Graph for the Initial State.

two steps its cost is 25 because it can be supported by drive(beta, gamma). The cost of the at(gamma) proposition is 25 for the rest of the planning graph, thus appearing with minimum cost in the last level. By knowing the minimum cost of each goal proposition it is possible to use max or sum heuristic to measure reachability cost. For example, the max heuristic for G is max($c(s_I,$ comm(soil)), $c(s_I,$ comm(rock)), $c(s_I,$ comm(image)))) = max(25, 40, 35) = 40, and the sum heuristic for G is 25 + 40 + 35 = 100. The set-level heuristic is no longer meaningful with cost functions because the level index is not directly related to cost.

It is possible to get an admissible cost-based heuristic when actions have nonuniform cost. The heuristic must never overestimate the aggregate cost of a set of propositions, meaning that both the cost of a set of precondition propositions and the goal propositions are aggregated by maximization. This corresponds to using max-propagation within a planning graph using ∞-lookahead and the max heuristic.

Cost-Sensitive Relaxed Plan Heuristics

Relaxed plan heuristics change in a significant way with cost functions. When there are several supporting actions for a proposition, it is best to bias the action choice toward one that supports with minimal cost (corresponding to the choice in line 7 of figure 4). It is not always enough to choose the action that is least costly; it is also important to consider the cost to support the preconditions of the action. The relaxed plan, as depicted in bold in figure 8, chooses supporting actions that have the least propagated cost among the possible supporters. For instance, comm(image) is supported at level 4 with the commun(image) action (cost 35) instead of the noop from level 3 (cost 40). Using propagated cost functions provides a look-ahead capability to the greedy extraction scheme that often results in low-cost relaxed plans. The relaxed plan heuristic is the sum of the execution costs of the actions in the action layers, providing 90 as the reachability heuristic for G. As mentioned earlier, extracting least cost relaxed plans is NP-hard.

Related Work

The Sapa planner (Do and Kambhampati 2003) introduced the ideas for cost propagation in metric-temporal planning problems. In metric-temporal planning problems it is common for planners to trade off the cost of a plan with its makespan (duration). We will discuss how to handle the temporal aspects of planning problems shortly. The "Hybrid Planning Graphs" section describes how temporal planning combines with cost-based planning.

The *POND* planner (Bryce and Kambhampati 2005) also makes use of cost propagation on a planning graph called the LUG (described in the nondeterministic planning section) for conditional planning problems. In conditional planning problems, the planner must often trade off the cost of sensing with the cost of acting. The ideas developed by our earlier work (Bryce and Kambhampati 2005) describe how to propagate costs over multiple planning graphs used for search in belief space.

Partial Satisfaction (Oversubscription) Planning

In the previous section we described handling action costs in the planning graph to guide search toward low-cost plans that achieve all of the goal propositions. However, as often happens in reality, the benefit of a plan that achieves all goals does not always justify the cost of the plan. In the rover example, the rover may expend considerable cost (power) in order to obtain an image while the benefit of having the image is low. However, it should get the image if the additional residual cost of obtaining the image is not too high. By balancing goal utility and plan cost one can satisfy a subset of the goals to maximize net benefit of the plan. Net benefit assumes that plan cost and goal utility are expressed in the same currency, such that net benefit is total goal utility minus plan cost.[7] The cost-sensitive heuristics discussed in the previous section can be applied to guide search toward plans with high net benefit by augmenting them to consider goal utility.

In the example, the utility function over the goal propositions is set such that comm(soil) has utility 50, comm(rock) has utility 60, and comm(image) has utility 20. We will assume an additive utility for the goals. Inspecting the optimal cost plans to achieve each subset of these goal propositions identifies that the optimal net benefit is 45 for the plan to achieve {comm(soil), comm(rock)}. Table 3 lists the associated cost, utility, and net benefit of the optimal plan for each goal subset.

Picking Goal Sets

The simplest way to handle partial satisfaction problems is to first pick the goal propositions that should be satisfied and then use them in a reformulated planning problem, as described by Van den Briel et al. (2004). The reformulated problem removes the goal utilities and retains action costs to find a cost-sensitive plan for the chosen subset of goal propositions. The trick is to decide which of the possible $2^{|G|}$ goal subsets to use in the reformulation. One way to reformulate the problem is to iteratively build a goal subset G' accounting for positive interactions between goals. Each iteration considers adding one proposition $g \in G \setminus G'$ to G' based on how it changes the estimated net benefit of G'.

The algorithm starts by initializing G′ with the most beneficial goal, by estimating the net benefit of each goal proposition with its utility minus its minimum propagated cost in the planning graph. In the example, the goal propositions have the following net benefits:

comm(soil) is utility 50 and cost 25 for net benefit 25,
comm(rock) is utility 60 and cost 40 for net benefit 20, and
comm(image) is utility 20 and cost 35 for net benefit –15.

The costs to achieve these goal propositions are, coincidentally, the same in both the planning graph (figure 8) and the true optimal plan in table 3, but this is not true in general. The algorithm chooses comm(soil) to initialize the goal set. Upon choosing the initial goal set G, the algorithm improves the estimate of its net benefit by extracting a relaxed plan (which will come in handy later). The relaxed plan for comm(soil) is:

$(\mathcal{A}_0^{RP}$ = {sample(soil, alpha)},
$\mathcal{A}_1^{RP}$ = {commun(soil)}),

which costs 25—the same as propagated cost.

After picking the initial goal proposition for the parent goal set G, the algorithm iterates through possible extensions of the goal set until the total net benefit stops increasing. The cost of each candidate extension is measured in terms of the cost of a relaxed plan to support the candidate goal. To measure as much positive interaction as possible, *the relaxed plan for the candidate goal is biased to reuse actions in the relaxed plan for the parent goal set.*

The candidate goal {comm(rock)} or the candidate goal {comm(image)} can be used to extend G'. The biased relaxed plan for {comm(soil), comm(rock)} is:

$(\mathcal{A}_0^{RP}$ = {sample(soil, alpha), drive(alpha, beta)},
$\mathcal{A}_1^{RP}$ = {commun(soil), sample(rock, beta)},
$\mathcal{A}_2^{R}$ = {commun(rock)}),

with utility 110 and cost 65 for net benefit 45. Similarly, {comm(soil), comm(image)} has the relaxed plan:

$$(\mathcal{A}_0^{RP} = \{sample(soil, alpha), drive(alpha, beta)\},$$
$$\mathcal{A}_1^{RP} = \{commun(soil), drive(beta, gamma)\},$$
$$\mathcal{A}_2^{RP} = \{sample(image, gamma)\},$$
$$\mathcal{A}_3^{R} = \{commun(image)\}),$$

with utility 70 and cost 60 for net benefit 10. The candidate goal {comm(rock)} increases net benefit the most (by 25), so it is added to G'.

The only extension of the new goal set is the entire set of goals {comm(soil), comm(rock), comm(image)}, which has the relaxed plan:

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

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

$$\mathcal{A}_2^{RP} = \{sample(image, gamma), commun(rock)\},$$
$$\mathcal{A}_3^{RP} = \{commun(image)\})$$

with utility 130 and cost 100 for net benefit 30. This decreases net benefit by 15 from 45, so the additional goal is not added. The reformulated planning problem for the chosen goal subset {comm(soil), comm(rock)} is solved by ignoring goal utility. Any of the cost-sensitive planning heuristics can be used to solve the subsequent problem. In order to guarantee an optimal net benefit plan, it may be necessary to find the optimal cost-sensitive plan for every goal subset and select the plan with highest net benefit.

Related Work

Partial satisfaction planning was first suggested as a generalization to classical planning by Smith (2002). The described techniques rely on the work of Van den Briel et al. (2004). Since this work, several algorithmic improvements and problem generalizations have appeared. The recent International Planning Competition (Gerevini, Bonet, and Givan 2006) included a track specifically to encourage and evaluate work on partial satisfaction planning.

One of the first extensions improves iterative goal set selection. Sanchez and Kambhampati (2005) develop methods for measuring negative interaction between goals in addition to the positive interaction described above. The approach involves using mutexes to measure the conflict between two goals and adjusting the heuristic measure of cost. This is another way of adjusting the heuristic to make up for relaxations as we discussed earlier.

Do, Benton, and Kambhampati (2006) discuss a generalization where the goal utilities have dependencies (for example, achieving comm(soil) and comm(rock) together is significantly more useful than achieving either in isolation). The ideas described above can be extended to handle these more expressive problems. Do, Benton, and Kambhampati (2006) use an integer linear program to improve the quality of relaxed plans by better accounting for utility dependencies.

An alternative to preselecting goal sets is to search for the best goal set (Do and Kambhampati 2004). Using progression search, it is possible to approximately determine when a plan suffix cannot improve net benefit. Using techniques similar to those just described, it is possible to estimate the remaining net benefit that is reachable in a possible plan suffix. Since it is possible to quickly find plans that satisfy some of the goals, an any-time search algorithm can return plans with increasing net benefit.

Another approach (Smith 2004) to oversubscription planning uses similar planning graph heuristics to derive an abstraction of the planning problem, called the orienteering problem. The solution to the orienteering problem provides a heuristic to guide search. Through a sensitivity analysis in the planning graph, through proposition and action removal, it is possible to identify propositions and actions that have the most influence on the cost of achieving goals, and these are used in the abstract problem.

Planning with Resources (Also Known as Numeric State Variables)

Planning with resources involves state variables that have integer and real values (also termed functions or numeric variables). Actions can, in addition to assigning a specific value to a variable, increment or decrement the value of a variable. Actions also have preconditions that, in addition to equality constraints, involve inequalities.

We augment our example, as shown in figure 9, to describe the rover battery power with the power function. Each action consumes power, except for a recharge action that allows the rover to produce battery power (through proper alignment of its solar panels). The rover starts with partial power reserves, that are not enough to achieve every goal. Thus, the rover needs a plan that is not much different from the classical formulation but adds some appropriately inserted recharge actions.

Planning graphs require a new generalization to deal with numeric variables. Previously, it was simple to represent the reachable values of variables because they were only Boolean valued. With numeric variables there is a possibly infinite number of reachable values. Depending on whether a variable is real or integer, there may be continuous or contiguous ranges of values that are reachable, which are

```
    (define (domain rovers_resource)                        (define (problem rovers_resource1)
     (:requirements :strips :typing)                         (:domain rovers_resource)
     (:types location data)                                  (:objects
     (:predicates (comm ?d - data)                                  soil image rock - data
                  (have ?d - data)                                  alpha beta gamma - location)
                  (at ?x - location)                         (:init (at alpha)
                  (avail ?d - data ?x - location))                  (avail soil alpha)
     (:functions (power)                                            (avail rock beta)
                 (effort ?x ?y - location)                          (avail image gamma)
                 (effort ?d - data))                                (= (effort alpha beta) 10)
     (:action drive                                                 (= (effort beta alpha) 5)
      :parameters (?x ?y - location)                                (= (effort alpha gamma) 30)
      :precondition (and (at ?x)                                    (= (effort gamma alpha) 5)
                    (>= (power) (effort ?x ?y)))                    (= (effort beta gamma) 15)
      :effect (and (at ?y) (not (at ?x))                            (= (effort gamma beta) 10)
                   (decrease (power) (effort ?x ?y))))              (= (effort soil) 20)
     (:action commun                                                (= (effort rock) 25)
      :parameters (?d - data)                                       (= (effort image) 5)
      :precondition (and (have ?d) (>= (power) 5))                  (= (power) 25))
      :effect (and (comm ?d) (decrease (power) 5)))         (:goal (and (comm soil)
     (:action sample                                                    (comm image)
      :parameters (?d - data ?x - location)                              (comm rock)))
      :precondition (and (at ?x) (avail ?d ?x)            )
                        (>= (power) (effort ?d)))
      :effect (and (have ?d) (decrease (power) (effort ?d))))
     (:action recharge
      :parameters ()
      :precondition (and)
      :effect (and (increase (power) 25)))
    )
```

Figure 9. PDDL Description of Resource Planning Formulation of the Rover Problem.

best represented as *intervals*. Tracking intervals in the planning graph helps determine when a particular enabling resource threshold can be crossed (for example, if the rover has enough power to perform an action). Because of the rich negative interactions surrounding resources, it is generally quite difficult to exactly characterize the resource intervals. The approach taken in many works is simply to represent the upper and lower bounds for possible values. Recall that the planning graph is optimistic in terms of reachability, so it is reasonable to represent only the upper and lower bounds even though some intermediate values are not in fact reachable.

Tracking Bounds

Consider the planning graph, shown in figure 10, for the initial state of the problem described in figure 9. The power variable is the only numeric variable whose value changes, and it is annotated with an interval of its possible values. In the first proposition layer, the possible values of the power variable are [25, 25], as given in the initial state. This makes only some of

the actions applicable in the first action layer; notice that drive(alpha, gamma) is not applicable because it requires 30 power units. The second proposition layer has the power bounds set to [−5, 50]. The lower bound is −5 because in the worst case all consumers are executed in parallel—sample(soil, alpha) consumes 20 units of power and drive(alpha, beta) consumes 10 units, so 25 − (10 + 20) = −5. Notice that determining the bounds on a numeric variable this way may provide loose bounds because many of the actions cannot really be executed in parallel. For instance, the lower bound on power in the second proposition layer is −5. Ignoring interactions between actions will prevent noticing that executing both sample(soil, alpha) and drive(alpha, beta) in parallel uses more power than is available (even though they individually have enough power to execute). Likewise, in the best case all producers execute in parallel: recharge as well as the persistence for power each produce 25 units of power, so 25 + 25 = 50.
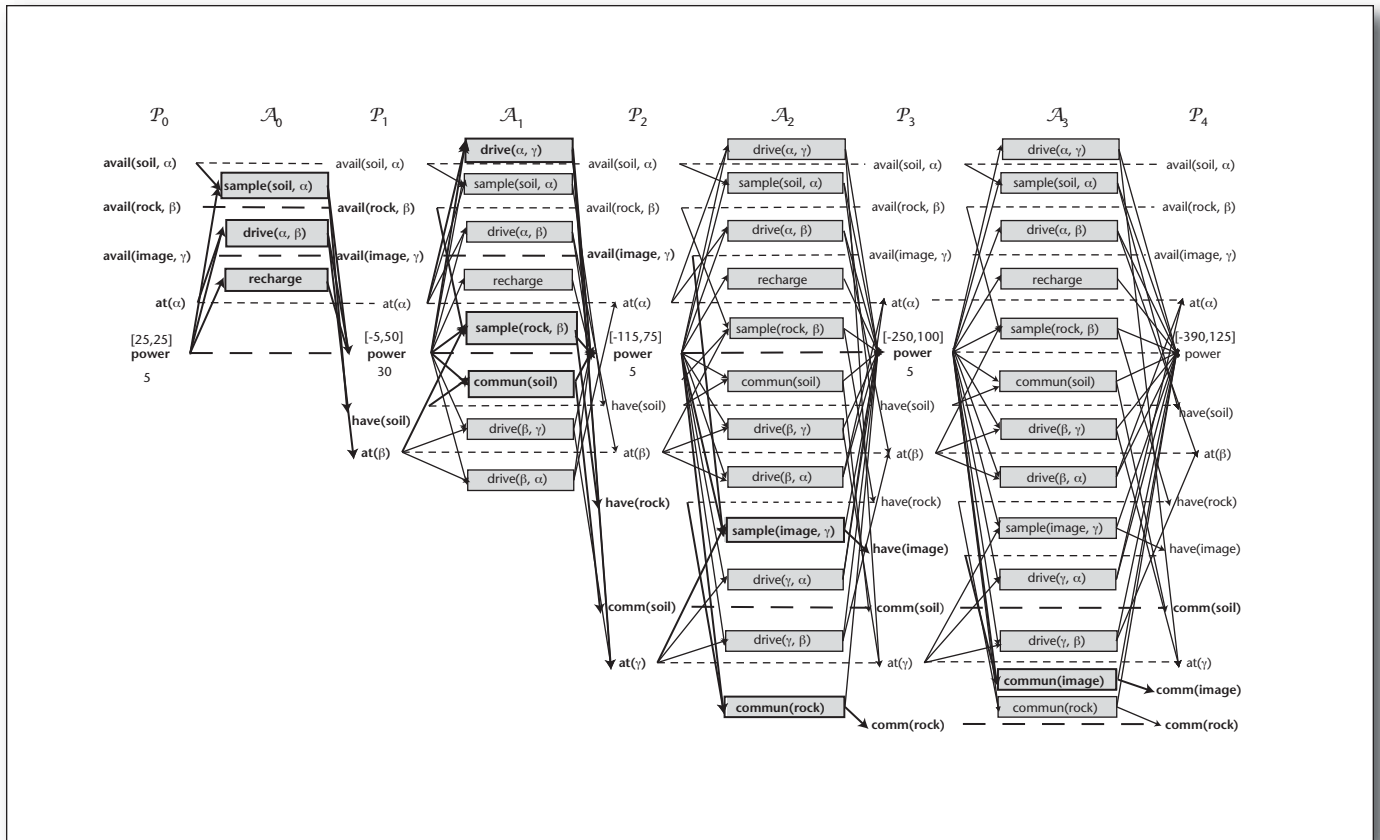
Figure 10. Resource Planning Graph for the Initial State.

Level-Based Heuristics

Level-based heuristics can be changed to accommodate resource intervals. The first level where a resource takes on a given value is the first level where the resource interval contains the value. For instance, if the goal were to have (> (power) 55), then the set-level heuristic estimates the cost to be 2, since power has the interval [–115, 75] in $\mathcal{P}_2$. As before, it is possible to aggregate the cost of several goal propositions to define a level-based heuristic.

Relaxed Plan Heuristics

Extracting a relaxed plan in the presence of numeric variables is a little more involved because *multiple supporters may be needed for each subgoal*.[8] A resource subgoal may need multiple supporters because its bounds are computed assuming that multiple actions produce or consume the resource. In the example, the power variable is a precondition of every action. Including an action in the relaxed plan requires that there be enough power. By ignoring interactions between actions, when several actions require power, the relaxed plan need only have enough power for the action that

requires the maximum amount of power. Consider the number listed below the power variable in figure 10. This number denotes the maximum amount of power needed to support an action chosen for the relaxed plan. In $\mathcal{P}_2^{RP}$ and $\mathcal{P}_3^{RP}$ the power requirement is 5 because the actions chosen for the respective $\mathcal{A}_2^{RP}$ and $\mathcal{A}_3^{RP}$ have a maximum power requirement of 5 (this includes persistence actions). The relaxed plan can support the respective power requirements by persistence from the previous proposition layer because the maximum power is greater than the requirements. In $\mathcal{P}_1^{RP}$, the power requirement is 30 because drive(alpha, gamma) in $\mathcal{A}_1^{RP}$ requires 30 units of power. Persistence alone cannot support the needed 30 units of power and the relaxed plan must use both the persistence of power and the recharge action to support the requirement.

Related Work

Planning graph heuristics for numeric state variables were first described by Hoffmann (2003), tracking only upper bounds. Sanchez and Mali (2003) describe an approach for tracking intervals. Benton, Do, and Kambhampati (2005) adapt planning graph heuristics to par-

tial satisfaction planning scenarios where the degree of satisfaction for numeric goals changes net benefit. The technique propagates costs for the resource bounds. As the bounds change over time, they track a cost for the current upper and lower bounds. Where cost propagation usually updates the cost of each value of a variable, with resources, the number of variable values is so large that only the costs of upper and lower bounds are updated.

Temporal Planning

To this point we have discussed planning models that have atomic time and are sequential—a single action is executed at each step. In temporal planning, actions have real duration and may execute concurrently. In the rover example (adapted from the classical model in figure 2 to have the durations depicted in figure 7), it should be possible for the rover to communicate data to the lander while it is driving.[9] Where the planning objective was previously plan length (or cost), with durative actions it becomes makespan (the total time to execute the plan). As such, temporal reachability heuristics need to estimate makespan. While temporal planning can have actions with cost, goal utility, and resources, we omit these features to concentrate solely on handling durative actions.

We adopt the conventions used by the Sapa planner (Do and Kambhampati 2003) to illustrate temporal reachability heuristics. Sapa constructs a temporal plan by adding actions to start at a given time. Sapa uses progression search, meaning that after adding some number of actions to start at time t, it will only add actions to time t' such that $t' > t$. The search involves two types of choices, either concurrently add a new action to start at time t (fatten) or advance the time t for action selection (advance). With concurrency and durative actions, the search state is no longer characterized by just a set of propositions. Each chosen action may have its effects occur at some time in the future. So the state is not just what is true now but also what the plan commits to making true in the future (delayed effects). With this extended state and temporal actions, temporal planning graph construction and heuristic extraction must explicitly handle time.

Temporal Planning Graph

Without durative actions the planning graph is constructed in terms of alternating proposition and action layers. The most important information is the "first level" where a proposition

is reachable (with this information and the action descriptions, it is possible to construct relaxed plans). In the temporal planning graph, heuristics are concerned with the "first time point" a proposition is reachable. Consider the temporal planning graph in figure 11 for the initial state. The temporal planning graph is the structure above the dashed line that resembles a Gantt chart. Each action is shown only once, denoting the first time it is executable and its duration. The propositions that are given as effects are denoted by the vertical lines; each proposition is shown only at the first time point it is reachable. The planning graph shows that comm(soil) is first reachable at time 11, comm(image) is first reachable at time 34, and comm(rock) is first reachable at time 76. In the temporal planning graph there is no explicit level because actions can span any duration and give effects at any time within their duration. In general, a durative action can have an effect at any time $t_a \leq t \leq t_a + dur(a)$, where t_a is the time the action executes, t is the time the effect occurs, and dur(a) is the action's duration. In the rover example, each action gives its effect at the end of its execution. For example, a = drive(alpha, gamma) starts at $t_a = 0$, and gives its positive effect at(gamma) at time $t_a + dur(a) = 0 + 30 = 30$. For an action to execute at a given time point t, it must be the case that all of its preconditions are reachable before t. In the example, sample(image, gamma) cannot execute until time 30 even though avail(image, gamma) is reached at time 0 because its other precondition proposition at(gamma) is not reachable until time 30. If the state includes past commitments to actions whose effects have not yet occurred (that is, the delayed effects discussed earlier), then the planning graph includes these actions with their start time possibly offset before the relative starting time of the planning graph.

It is also possible to compute mutexes in the temporal planning graph, but they require generalization to deal with concurrent actions that have different durations. Following Temporal GraphPlan (Smith and Weld 1999), in addition to tracking proposition/proposition and action/action mutexes, it becomes necessary to find proposition/action mutexes explicitly. While we refer the reader to Smith and Weld (1999) for a complete account, proposition/action mutexes capture when it is impossible for a proposition to hold in any state while a given action executes. In nontemporal planning, these mutexes were captured by mutexes between actions and noops.
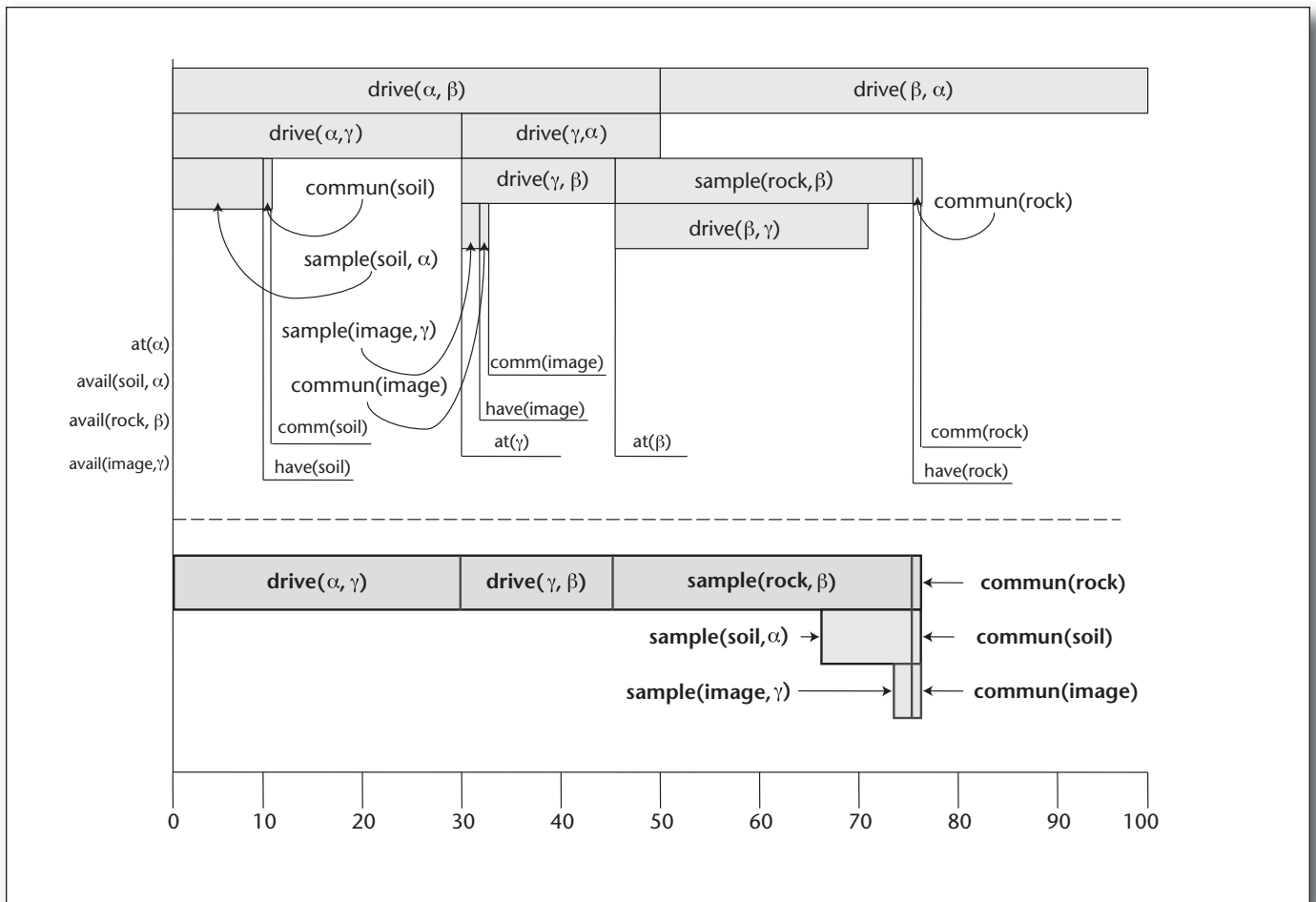
Figure 11. Temporal Planning Graph (top) and Relaxed Plan (bottom) for the Initial State.

Temporal Planning Graph Heuristics

The temporal planning graph for the initial state of the rover example shows that every goal proposition is reached by time 76, providing a lower bound on the makespan of a valid plan. This is an estimate of makespan that is similar to the set-level heuristic in classical planning. Instead of finding the level where the goal propositions are reachable, the heuristic finds the time point where they are reachable. Analogous to their role in classical planning, mutexes can help improve makespan estimates by reducing optimism in the planning graph.

In addition to makespan estimates, it is often convenient to use temporal relaxed plans to estimate the plan cost. Similar to level-based heuristics, relaxed plans use time points in the place of levels. Recall that a classical relaxed plan is a set of propositions and actions that are indexed by the level where they appear. Temporal relaxed plans use a similar scheme but index the propositions and actions by time.

Extracting a temporal relaxed plan involves starting from the time point where all goals are reachable and recursively supporting each goal as shown in figure 4. The main difference is that instead of decrementing levels across the recursions, it decrements the time by which subgoals are needed. Figure 11 shows the relaxed plan in bold below the dashed line. It supports all of the goals at time 76. To support a proposition at a time *t,* the relaxed plan chooses an action *a* and sets its start time to *t – dur(a)* (that is, the action starts as late as possible). For instance, commun(image) is used to support comm(image) at time 76 by starting at time 75. Each chosen action has its precondition propositions supported at the time the action starts. The commun(image) action starts at time 75, so its precondition proposition have(image) must be supported by time 75. If the state includes commitments to actions with delayed effects, then relaxed plan extraction can bias or force selection of such actions to support propositions.

Related Work

Temporal GraphPlan (Smith and Weld 1999) was the first work to generalize planning graphs to temporal planning, while Sapa (Do and Kambhampati 2003) was the first to extract heuristics from this type of planning graph.

There are many extensions of GraphPlan to temporal planning. The temporal planning graph, as we and Smith and Weld (1999) have presented, is implemented with unique datastructure called a bilevel planning graph (Long and Fox 1999; Smith and Weld 1999). The bilevel planning graph is a simple data structure that has two layers: an action layer and a proposition layer, which are connected with arcs corresponding to preconditions and effects. Bilevel planning graph expansion involves propagating lower bounds on the times to reach propositions and actions without physically generating extra levels. Bilevel planning graphs can also be used in nontemporal planning to save memory.

TPSys (Garrido, Onaindia, and Barber 2001) is another GraphPlan-like planner that uses a different generalization of planning graphs in temporal planning. Instead of removing all notion of level to accommodate actions with different durations, TPSys associates time stamps with proposition layers and allows durative actions to span several levels. For example, the proposition layer after one step is indexed by the end time of the shortest duration action, and it contains this action's effects. The explicit proposition layers can aid mutex propagation, allowing TPSys to compute more mutexes than Temporal GraphPlan (Garrido, Onaindia, and Barber 2001). The TPSys planning graph has not yet been used for computing reachability heuristics.

Another approach for using GraphPlan in temporal planning, in the LPGP planner (Long and Fox 2003), uses a classical planning graph to maintain logical reachability information and a set of linear constraints to manage time. By decoupling the planning graph and time, LPGP supports a richer model of durative actions. The manner in which LPGP uses the planning graph, for plan search, has not yet been extended to computing reachability heuristics.

Many of the techniques discussed in classical planning show up in temporal planning. For instance, Sapa adjusts its heuristic to account for negative interactions by using mutexes to reschedule its temporal plan to minimize conflicts, resulting in a better makespan estimate. Sapa also uses cost functions to estimate cost in addition to makespan.

The LPG planner (Gerevini, Saetti, and Serina 2003) generalizes from classical planning by using a temporal planning graph (most similar to Temporal GraphPlan) to perform local search in the planning graph. LPG uses temporal relaxed plan heuristics, similar to those described above, to guide its search.

Another interesting and recent addition to PDDL allows for timed initial literals. These are propositions that will become true at a specified time point (a type of exogenous event). Integrating timed initial literals into the temporal planning graph is easy because there are implicit exogenous actions that give the literal as an effect. However, heuristics change considerably because the planner no longer has direct choice over every state it reaches. Many of the existing techniques to handle timed initial literals adjust relaxed plans with common scheduling heuristics (Gerevini, Saetti, and Serina 2005)—another example of adjusting heuristics to make up for relaxation.

Nondeterministic Planning

None of the planning models we have considered to this point allow uncertainty. Uncertainty can arise in multiple ways: a partially observable state, noisy sensors, uncertain action effects, or uncertain action durations. We first concentrate on the simplest uncertainty, an unobservable and uncertain state. This falls into the realm of nondeterministic conformant planning with deterministic actions (discussed later). Later in this section, we discuss how the nondeterministic conformant planning heuristics extend to conditional (sensory) planning. In the next section we consider stochastic planning with uncertain actions.

Background

The conformant planning problem starts with an uncertain state and a set of actions with conditional effects. The uncertain state is characterized by a belief state b (a set of possible states), such that $b \subseteq 2^P$. Actions with conditional effects are common in conformant planning because the executed actions must be applicable to all states in the belief state. Instead of requiring strict execution preconditions, it is common to move the conditions into secondary preconditions that enable conditional effects. This way actions are executable in belief states, but their effect depends on the true state in the belief state. Since the true state is not known, the plan becomes conformant (that is, the plan should work no matter which state in the belief state is the true state).

Consider a modification to the rover exam-

```
(define (domain rovers_conformant)          (define (problem rovers_conformant1)
 (:requirements :strips :typing)             (:domain rovers)
 (:types location data)                      (:objects
 (:predicates                                      soil image rock - data
  (at ?x - location)                               alpha beta gamma - location)
  (avail ?d - data ?x - location)           (:init (at alpha)
  (comm ?d - data)                                 (oneof (avail soil alpha)
  (have ?d - data))                                (avail soil beta)
 (:action drive                                    (avail soil gamma)))
  :parameters (?x ?y - location)            (:goal (comm soil))
  :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 (at ?x)
  :effect (when (avail ?d ?x) (have ?d)))
)
```

Figure 12. PDDL Description of Conformant Planning Formulation of Rover Problem.

ple, listed in figure 12. The sample action has a conditional effect, so that sampling works only if a sample is available; otherwise it has no effect. The initial state is uncertain because it is unknown where the rover can obtain a soil sample (the terrain is quite hard and rocky), but it is available at one of locations alpha, beta, or gamma. The initial belief state is defined:

$$b_I = \{\{at(alpha), avail(soil, alpha)\},$$
$$\{at(alpha), avail(soil, beta)\},$$
$$\{at(alpha), avail(soil, gamma)\}\}.$$

To simplify the example, the only goal proposition is comm(soil). At this point, the rover cannot determine whether it has the soil sample because its sensor has broken. The solution to this problem is to go to each location and sample to be sure the rover has the soil sample, and then communicate the data back to the lander:

{sample(soil, alpha), drive(alpha, beta),
 sample(soil, beta), drive(beta, gamma),
 sample(soil, gamma), commun(soil)},

resulting in the belief state:

{{at(gamma), avail(soil, alpha), have(soil),
  comm(soil)},
 {at(gamma), avail(soil, beta), have(soil),
  comm(soil)},

{at(gamma), avail(soil, gamma), have(soil),
  comm(soil)}},

where the goal comm(soil) is satisfied in *every* possible state.

To clarify the semantics for applying actions to belief states, consider applying sample(soil, alpha) to b_I. The action is applicable because its precondition at(alpha) is satisfied by every state in the belief state. The resulting belief state b' is the union of the successors obtained by applying the action to every state in b_I:

$$b' = \{\{at(alpha), avail(soil, alpha), have(soil)\},$$
$$\{at(alpha), avail(soil, beta)\},$$
$$\{at(alpha), avail(soil, gamma)\}\}.$$

The first state is the only state that changes because it is the only state where the conditional effect is enabled. Applying the drive(alpha, beta) action to b results in:

$$b'' = \{\{at(beta), avail(soil, alpha), have(soil)\},$$
$$\{at(beta), avail(soil, beta)\},$$
$$\{at(beta), avail(soil, gamma)\}\}.$$

Every state in b'' changes because the action applies to every state.

Single Planning Graphs

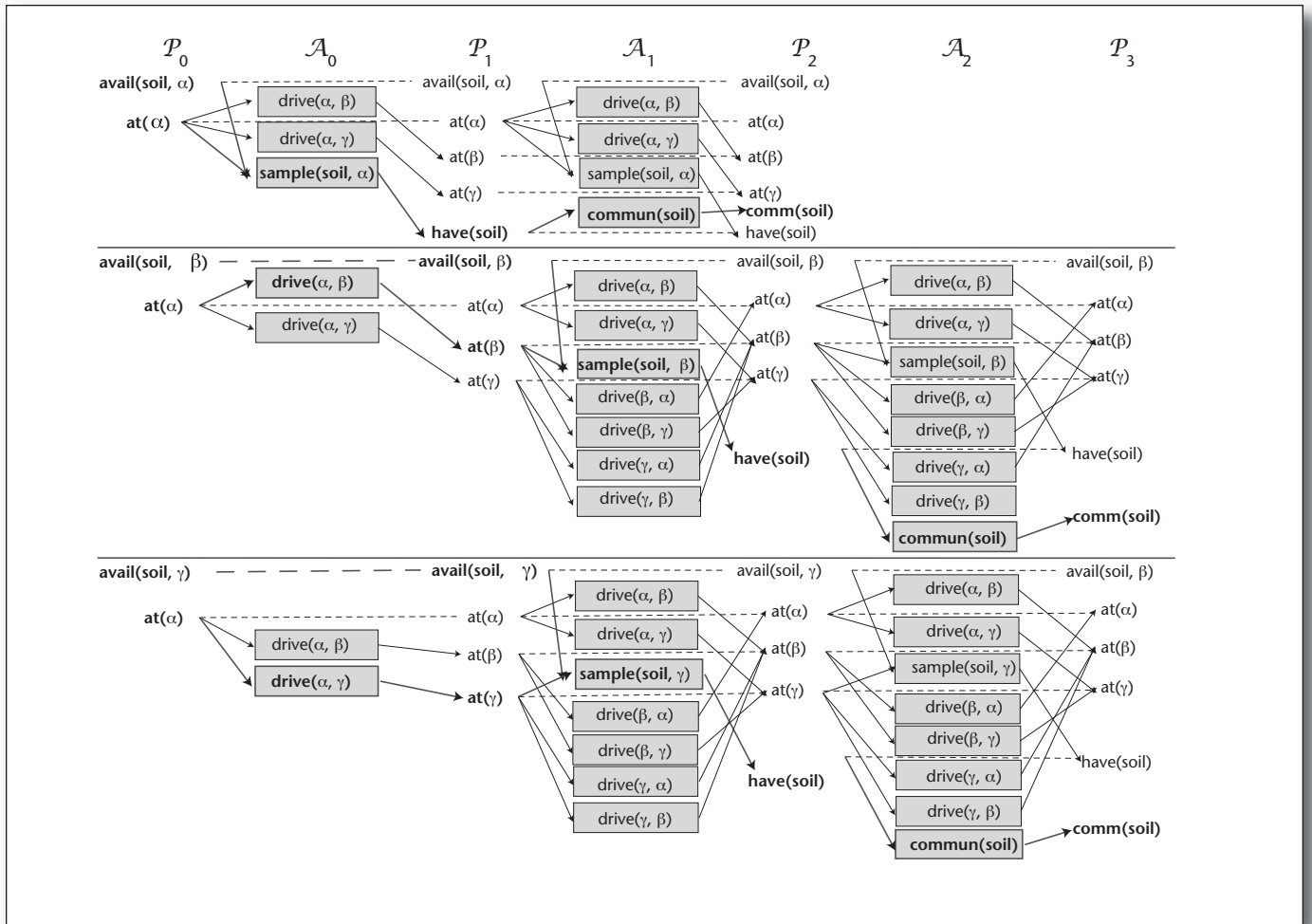There are many ways to use planning graphs

Figure 13. Conformant Planning Graphs.

for conformant planning (Bryce, Kambhampati, and Smith 2006a). The most straightforward approach is to ignore state uncertainty. There is nothing about classical planning graphs that requires defining the first proposition layer as a state; every other proposition layer already represents a set possibly reachable of states. One could union the propositions in all states of the belief state to create an initial proposition layer, such as:

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

for the initial belief state. The planning graph built from the unioned proposition layer has the goal appear after two levels, and the relaxed plan would be:

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

which gives 2 as the heuristic (greatly underestimating plan cost). Alternatively, one could sample a single state from the belief state to use for the planning graph (which may also lead to

an underestimate). Using either of these approaches results in a heuristic that measures the cost to reach the goal from one (or some intersection) of the states in a belief state. This is typically an underestimate because a conformant plan must reach the goal from *all* of the states in a belief state.

Multiple Planning Graphs

A more systematic approach to using planning graphs involves constructing a planning graph for each state in the belief state, extracting a heuristic estimate from each planning graph, and aggregating the heuristics. Similar to the level-based heuristics in classical planning that aggregate proposition costs, it is possible to aggregate the heuristic for each possible state by assuming full positive interaction (maximization) or full independence (summation) between the states. In the example there is positive interaction between the possible initial states because they can all benefit from using

the commun action at the end of the plan. There is also independence between the initial states because sampling works only at one location for each. Relaxed plans that do not simply numerically aggregate individual relaxed plan heuristics can capture (in)dependencies in the heuristic for the belief state, as was possible for goal propositions in classical planning. Numeric aggregation loses information about positively interacting and independent casual structure that is used in common across the relaxed plans. Taking the union of the relaxed plans to get a unioned relaxed plan will better measure these dependencies.

To see how to capture both positive interaction and independence between states in a belief state, consider figure 13. We illustrate the planning graph for each state in the initial belief state and a relaxed plan (in bold) to support the goal. There are three relaxed plans that must be aggregated to get a heuristic measure for the initial belief state b_I. Using the maximum cost of the relaxed plans gives 3 as the heuristic, and using the summation of costs gives 8 as the heuristic. Neither of these is very accurate because the optimal plan length is 6. Instead, one can step-wise union the relaxed plans. The intuition is that if an action appears in the same layer for several of the relaxed plans, then it is likely to be useful for several states in the belief state, meaning the plans for states positively interact. Likewise, actions that do not appear in multiple relaxed plans in the same layer are used by the states independently. Taking the step-wise union of the relaxed plans gives the unioned relaxed plan:

$$(\mathcal{A}_0^{RP} = \{\text{sample(soil, alpha), drive(alpha, beta),}$$
$$\text{drive(alpha, gamma)}\},$$
$$\mathcal{A}_1^{RP} = \{\text{commun(soil), sample(soil, beta),}$$
$$\text{sample(soil, gamma)}\},$$
$$\mathcal{A}_2^{RP} = \{\text{commun(soil)}\}).$$

The unioned relaxed plan contains seven actions, which is closer to the optimal plan length. The unioned relaxed plan can be poor when the individual relaxed plans are not well aligned, that is, they contain many common actions that appear in different layers. In the next approach, it is possible to overcome this limitation.

Labeled Planning Graphs

While the multiple planning graph approach to obtain unioned relaxed plans is informed, it has two problems. First, it requires multiple planning graphs, which can be quite costly when there are several states in the belief state; plus, there is a lot of repeated planning graph structure among the multiple planning graphs. Second, the relaxed plans used to obtain the

unioned relaxed plan were extracted independently and do not actively exploit positive interactions (like we have seen in previous sections). For example, the unioned relaxed plan has commun(soil) in two different action layers where, intuitively, it is needed to execute only once.

The solution to both these problems is addressed with the labeled (uncertainty) planning graph (LUG). The LUG represents multiple explicit planning graphs implicitly. Figure 14 shows the labeled planning graph representation of the multiple planning graphs in figure 13. The idea is to use a single planning graph skeleton to represent proposition and action connectivity and to use labels to denote which portions of the skeleton are used by which of the explicit multiple planning graphs. Each label is a propositional formula whose models correspond to the propositions used in the initial layers of the explicit planning graphs. In figure 14, there is a set of shaded circles above each proposition and action to represent labels (ignore the variations in circle size for now). The black circles represent the planning graph for the first state {at(alpha), avail(soil, alpha)} in b_I, the grey circles are for the state {at(alpha), avail(soil, beta)}, and the white circles are for the state {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.

The LUG is constructed by propagating labels (logical formulas) a layer at a time. The intuition behind label propagation is that actions are applicable in every explicit planning graph where their preconditions are supported, and that propositions are supported in every explicit planning graph where they are given as an effect. For example, sample(soil, alpha) in $\mathcal{A}_0$ is labeled only with black to indicate it has an effect in the explicit planning graph only for state {at(alpha), avail(soil, alpha)} (inspecting the explicit planning graphs in figure 13 will confirm this). Similarly, have(soil) is supported in $\mathcal{P}_1$ only for the state {at(alpha), avail(soil, alpha)}. More formally, if the labels refer to sets of states (but are represented by Boolean formulas), then an action is labeled with the intersection (conjunction) of its preconditions' labels, and a proposition is labeled with the union (disjunction) of its supporters' labels. The goal is reachable in the LUG when every goal proposition has a label with every state in the source belief state. In the example, the goal comm(soil) is first reached by a single source state {at(alpha), avail(soil, alpha)} in $\mathcal{P}_2$ and is reached by every state in $\mathcal{P}_3$.
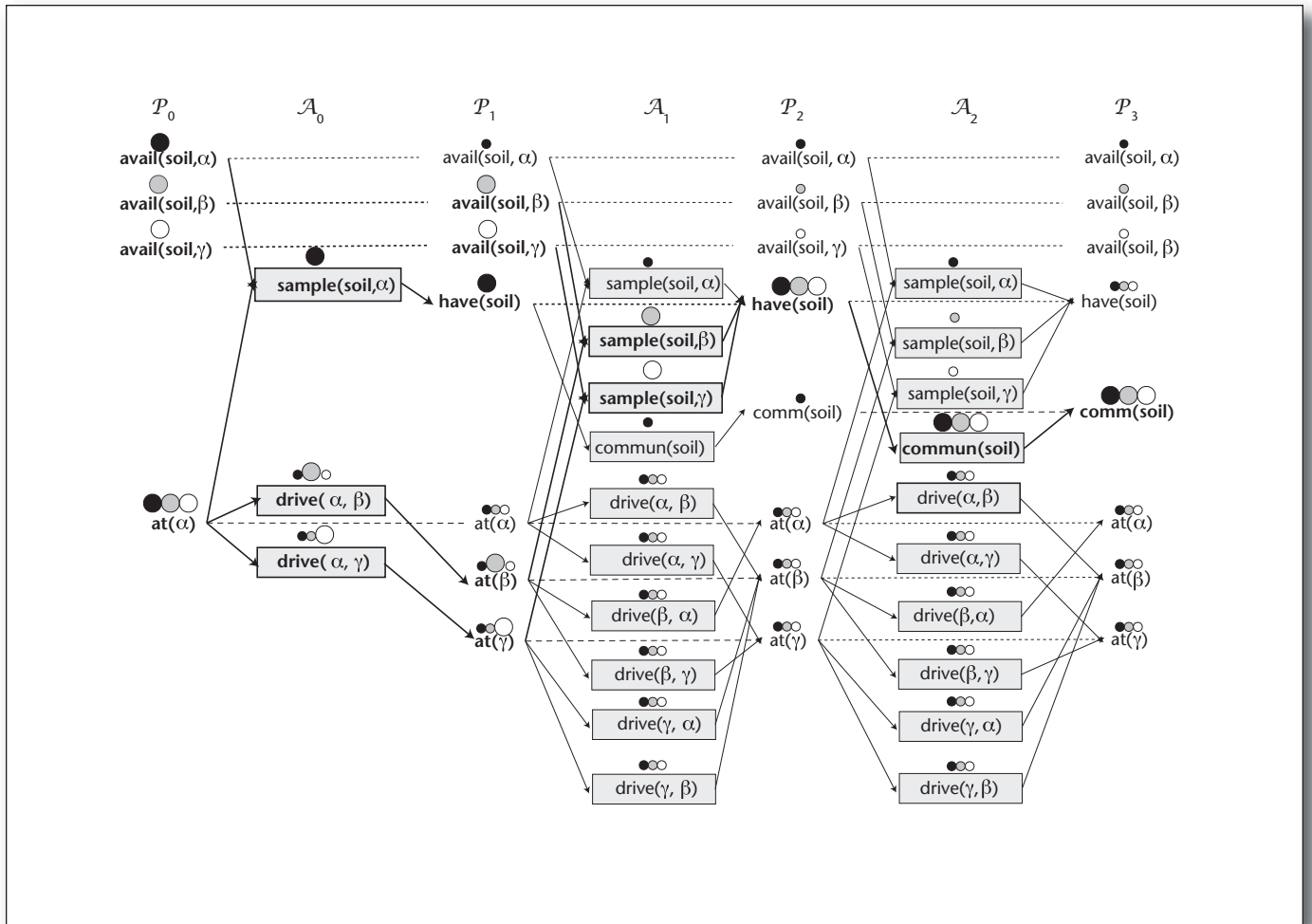
Figure 14. Conformant Labeled Uncertainty Graph.

Labeled Relaxed Plans

With a succinct, implicit representation of the multiple planning graphs it is possible to extract a relaxed plan. From the LUG one can extract the relaxed plan for all states simultaneously and exploit positive interaction. 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* (such as when using resources in the planning with resources section) *because it needs support from several source states, each requiring a supporter.* Like relaxed plans for resources, there is some additional bookkeeping needed for the labeled relaxed plan. Specifically, the relaxed plan tracks which of the source states require the inclusion of a particular action or proposition. Notice the circles in figure 14 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 $\mathcal{P}_1^{RP}$ is used only for the

source state {at(alpha), avail(soil, gamma)} (the white circle). 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 $\mathcal{P}_2^{RP}$ 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 $\mathcal{P}_2^{RP}$ with supporters, the relaxed plan selects the noop for have(soil) for support from state {{at(alpha), avail(soil, alpha)} (the black circle), sample(soil, beta) for support from state {at(alpha), avail(soil, beta)} (the gray circle), and sample(soil, alpha) for support from state {at(alpha), avail(soil, gamma)} (the white circle). 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 4): an action that supports in several of the explicit planning graphs

is better than an action that supports in only one. Notice that the labeled relaxed plan contains the commun(soil) action only once as opposed to twice in the unioned relaxed plan. The labeled relaxed plan gives a heuristic measure of 6, which is the optimal cost of the real plan.

Related Work

Our discussion is based our previous work on planning graph heuristics for nondeterministic planning (Bryce, Kambhampati, and Smith 2006a; Bryce and Kambhampati 2004).

Another approach is developed in the Conformant and Contingent FF planners (Hoffmann and Brafman 2004; Brafman and Hoffmann 2005). Both planners do not explicitly represent belief states; rather they keep track of the initial belief state and an action history to characterize plan prefixes. From a set of propositions that must hold at the end of the action history, a planning graph is constructed. The initial belief state, the action history, and the planning graph are translated to a satisfiability instance. The solution to the satisfiability problem identifies a relaxed sequence of actions that will make the goal reachable. Rather than directly propagating belief state information over the planning graph, as with the labeled planning graph, this technique allows the satisfiability solver to propagate belief state information from the initial belief state.

A further extension of the LUG has been used to develop the state agnostic planning graph (SAG) (Cushing and Bryce 2005). We previously mentioned that planning graphs are single source and multiple destination structures. The SAG is a multiple source planning graph, which like the LUG represents multiple planning graphs implicitly. The SAG represents every planning graph that progression search will require, making heuristic evaluation at every search node amortize the cost of constructing the SAG.

The GPT planner (Bonet and Geffner 2000b) and the work of Rintanen (2004) offer alternative reachability heuristics for nondeterministic planning. GPT relies on a relaxation of nondeterministic planning to full observability. For instance, GPT measures the heuristic for a belief state to reach a goal by finding the optimal deterministic plan for each state in the belief state. The maximum cost deterministic plan among the states in the belief state is a lower bound on the cost of a nondeterministic plan. Rintanen (2004) generalizes the GPT heuristic by finding the optimal nondeterministic plan for every size n subset of states in a belief state. The maximum cost nondetermin-

istic plan for a size n set of states is a better lower bound on the optimal nondeterministic plan for the belief state.

It is possible to use the conformant planning heuristics, described above, to guide search for conditional (sensory) plans. Conditional plans branch based on execution time feedback because of negative interactions. If, for example, the sample action will break the sampling tool if no soil sample is available (because the ground is otherwise rocky), then the rover must sense before sampling. Breaking the sampling tool will introduce a negative interaction that prevents finding a conformant plan. Creating a plan branch for whether the location is rocky can avoid breaking the tool (and the negative interaction it introduces). Since many heuristics work by ignoring negative interactions, conformant planning heuristics will work by ignoring the observations and the negative interactions they remove. This is the approach taken by Bryce, Kambhampati, and Smith (2006a) and Brafman and Hoffmann (2005) to a reasonable degree of success. The reason using conformant relaxed plans is not overly detrimental is that while search may need to generate several conditional plan branches, the total search effort over all such branches is related to the number of actions in the relaxed plan. The conformant relaxed plan simply abstains from separating the chosen actions into conditional branches. The benefit of further incorporating negative interactions and observations into conditional planning heuristics is currently an open issue.

Stochastic Planning

Stochastic planning characterizes a belief state with a probability distribution over states and the uncertain outcomes of an action with a probability distribution. Consider an extension of the conformant version of the rover example to contain probability and uncertain actions, in figure 15. The sample and commun actions have an uncertain outcome—they have the desired effect or no effect at all. In the general case, there may be multiple different outcomes. The initial state is uncertain, as before, but is now characterized by a probability distribution. Plans for this formulation will require a number of repetitions of certain actions in order to raise the probability of satisfying the goal (which is required to be at least 0.5 in the problem). For example, executing sample(soil, alpha), followed by commun(soil), will result in only one state in the belief state satisfying the goal comm(soil) (its probability is $0.4(0.9)0.8 = 0.29$). However, executing an

```
   (define (domain rovers_stochastic)            (define (problem rovers_stochastic1)
    (:requirements :strips :typing)               (:domain rovers)
    (:types location data)                        (:objects
    (:predicates                                        soil image rock - data
     (at ?x - location)                                 alpha beta gamma - location)
     (avail ?d - data ?x - location)             (:init (at alpha)
     (comm ?d - data)                                   (probabilistic 0.4 (avail soil alpha)
     (have ?d - data))                                               0.5 (avail soil beta)
    (:action drive                                                   0.1 (avail soil gamma)))
     :parameters (?x ?y - location)              (:goal (comm soil) 0.5)
     :precondition (at ?x)                       )
     :effect (and (at ?y) (not (at ?x))))
    (:action commun
     :parameters (?d - data)
     :precondition (have ?d)
     :effect (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))))
   )
```

Figure 15. PDDL Description of Stochastic Planning Formulation of the Rover Problem.

additional commun(soil) raises the probability of the state where comm(soil) holds to 0.4(0.9)(0.8 + 0.8(0.2)) = 0.35. By only performing actions at alpha, the maximum probability of satisfying the goal is 0.4, so the plan should also perform some sampling at beta.

Exact LUG Representation

It is possible to extend the LUG to handle probabilities by associating probabilities with each label. However, handling uncertain actions, whether nondeterministic or stochastic, is troublesome. With deterministic actions, labels capture uncertainty only about the source belief state, and the size of the labels is bounded (there are only so many states in the belief state). With uncertain actions, 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. In the LUG with uncertain actions, labels can get quite large and costly to propagate for the purpose of heuristics.

Monte Carlo in Planning Graphs

One does not need to propagate labels for every uncertain action outcome and possible state because, as seen in the previous section, there is usually considerable positive interaction in the LUG. In order to include an action in the relaxed plan, one need not know every uncertain state or action that causally supports the action, just the most probable. Thus, it becomes possible to sample which uncertain states and action outcomes enter the planning graph. While it is possible to use a single (unlabeled) planning graph by sampling only one state and outcome of each action, using several samples better captures the probability distributions over states and action outcomes encountered in a conformant plan. Each sample can be thought of as a deterministic planning graph.

The Monte Carlo LUG ($\mathcal{M}c$LUG) represents every planning graph sample simultaneously using the labeling technique developed in the LUG. Figure 16 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 edge is labeled by particles (sometimes with fewer particles than the associated action). The $\mathcal{M}c$LUG can be thought of as sequential Monte Carlo (SMC) (Doucet, de Fre-
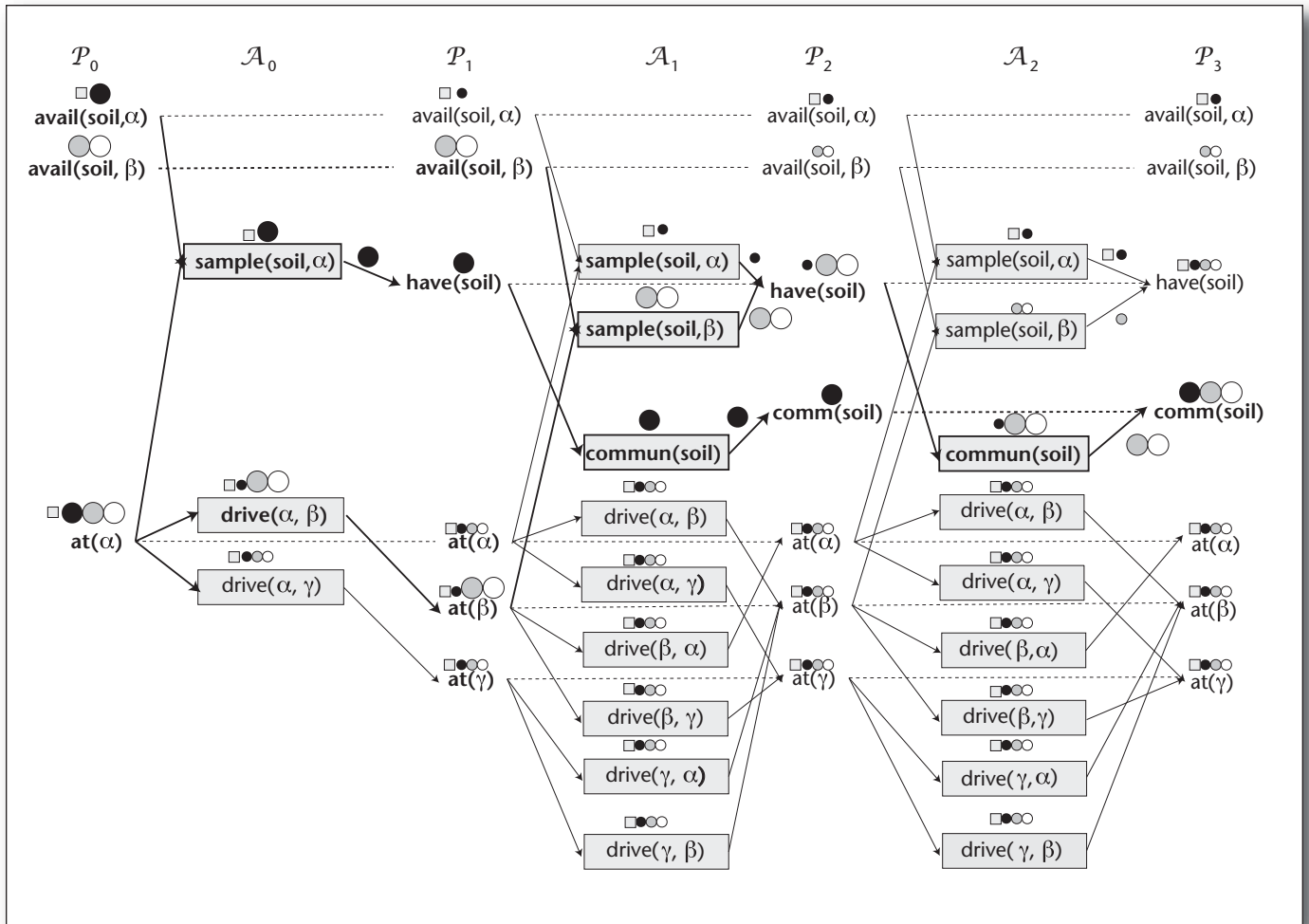
Figure 16. Monte Carlo Labeled Uncertainty Graph.

itas, and Gordon 2001) in the relaxed planning space. SMC is a technique used in particle filters for approximating the posterior probability of a random variable that changes over time. The idea is to draw several samples from a prior distribution (a belief state) and then simulate each sample through a transition function (an action layer). Where particle filters use observations to weight particles, the *Mc*LUG does not.

Returning to figure 16, there are four particles, the first two sampling the first state in the belief state, and the latter two sampling the second state. The third state, 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 its effect 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 *Mc*LUG must resample which particles label each effect 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. The labeled relaxed plan contains commun(soil) twice, reflecting the need to repeat actions that may fail to give their desired effect. The commun(soil) action is used twice because in $\mathcal{A}_2$ the black particle does not sample the desired outcome. The black particle must support

comm(soil) through persistence from the previous level, where it was supported by the desired outcome of commun(soil). A plan using the actions identified by the relaxed plan could satisfy the goal with probability (0.4 (0.9) + 0.5 (0.9)) (0.8 + (0.2) 0.8) = 0.78, which exceeds the probability threshold. Notice also that 0.78 is close to the number of particles (3/4) reaching the goal in the $McLUG$.

Related Work

The discussion of the $McLUG$ is based on the work of Bryce, Kambhampati, and Smith (2006b).

The work on Conformant FF extends to the probabilistic setting in Probabilistic FF (Domshalk and Hoffmann 2006). Previously the planner relied on SAT, and now it uses weighted model counting in CNFs. The relaxed plan heuristic is found in a similar fashion, using a weighted version of the planning graph.

Where the $McLUG$ uses labels and simulation to estimate the probability of reaching the goals, it is possible to directly propagate probability on the planning graph. Propagating a probability for each proposition in order to estimate the probability of a set of propositions greatly underestimates the probability of the set. The technique developed by Bryce and Smith (2006) propagates a binary interaction factor $I(a, b)$ that measures the positive or negative interaction between two propositions, actions, or effects. After computing $I(a, b) = Pr(a \wedge b) / Pr(a)P(b)$, having $I(a, b) = 0$ means that a and b are mutex, having $I(a, b) = 1$ means that a and b are independent, having $0 < I(a, b) < 1$ means that a and b negatively interact, and having $1 < I(a, b)$ means that a and b positively interact. Consequently, the measure of interaction can be seen as a continuous mutex. Propagating interaction can benefit nonprobabilistic planning also, for example, by defining it in terms of cost $I(a, b) = c(a, b) - (c(a) + c(b))$.

Hybrid Planning Graphs

In many of the previous sections we largely concentrated on how to adapt planning graphs to handle a single additional feature over and above classical planning. In this section we discuss a few works that have combined these methods to handle more expressive planning problems. The problems are metric-temporal planning, partial satisfaction planning with resources, temporal planning with uncertain actions, and cost-based nondeterministic planning.

Metric-Temporal Planning

In metric-temporal planning, the problem is multiobjective because the planner must find a plan with low makespan using nonuniform cost actions. For instance, the rover may drive at two different speeds, the faster speed costing more than the slower. The rover can drive fast and achieve the goals very quickly, but at high cost.

Recall from the "Temporal Planning" section that the measure of makespan is identical with level-based and relaxed plan heuristics. In metric-temporal planning, level-based heuristics measure makespan and relaxed plans measure plan cost. To obtain a high-quality relaxed plan when the actions have nonuniform cost, a planning graph can propagate cost functions over time (instead of levels) (Do and Kambhampati 2003). Instead of updating the cost of propositions or actions at every time point, it updates only when they receive a new supporter or the cost of a supporter drops. Then, instead of supporting the goals when they are first coreachable, the relaxed plan supports them at the time where they have minimum cost. With a heuristic measure of makespan and cost, a user-supplied combination function can help guide search toward plans matching the user's preference.

Partial Satisfaction with Resources

In the "Planning with Resources" section we mentioned planning for different degrees of satisfaction for numeric goals (Benton, Do, and Kambhampati 2005). The techniques for selecting goals for a reformulated planning problem can be adapted to numeric goals. First, the planning graphs need to propagate cost functions for resources, and second, the relaxed plan must decide at what resource level to satisfy the (numeric) resource goals to maximize net benefit.

Since the planning graph propagates resources in terms of upper and lower bounds, it is easy to capture the reachable values of a resource. However, the cost of each value of a resource is much more difficult to propagate. An exact cost function for all resource values in the reachable interval can be nonlinear in general. The approach taken by Benton, Do, and Kambhampati (2005) is to track a cost for only the upper and lower bounds of the resource. The bound that enables an action is used to define the cost of supporting the action. In order to decide which level to satisfy a numeric goal, the relaxed plan can use the propagated cost of the resource bounds or relaxed plans to find the maximal net benefit value.

Temporal Planning with Uncertainty

The Prottle planner (Little, Aberdeen, and Thiebaux 2005) addresses temporal planning with uncertain actions whose durations are also uncertain. Unlike the techniques for partially observable problems described in the nondeterministic planning section, Prottle finds conditional plans for fully observable problems (so there is no need to reason about belief states). Prottle uses a planning graph to provide a lower bound on the probability of achieving the goal. The idea is to include an extra layer at each level to capture the uncertain outcomes of actions. Prottle back-propagates the probability of reaching each goal proposition on the planning graph to each other proposition. To define the heuristic merit of each state, Prottle uses the aggregate probability that each proposition reaches the goal propositions. By using back-propagation, Prottle is able to avoid building a planning graph for each search node, but it must be able to identify which planning graph elements to use in the heuristic computation.

Cost-Based Conditional Planning

In our previous work (Bryce and Kambhampati 2005), we developed a cost-based version of the LUG, called the *C*LUG. The *C*LUG propagates costs on the LUG to extract labeled relaxed plans that are aware of action costs. The key insight of the *C*LUG is to not propagate cost for every one of the implicit planning graphs (of which there may be an exponential number). Rather, planning graphs are grouped together and the propositions and actions they have in common are assigned the same cost. With costs, it is possible to extract labeled relaxed plans that bias the selection of actions.

Conclusion and Discussion

We have presented the foundations for using planning graphs to derive reachability heuristics for planning. These techniques have enabled many planning systems with impressive scalability. We have also shown several extensions to the classical planning model that rely on many of the same fundamental methods.

It is instructive to understand the broad reasons why planning graph heuristics have proven to be so widely useful. Reachability heuristics based on planning graphs are useful because they are forgiving, they can propagate multiple types of information, they support phased relaxation, they synergize with other planner features, and above all they are versatile. By forgiving, we mean that we can pick and choose the features to compute (for example, mutexes). As we saw, the planning graph can propagate all types of information, such as levels, subgoal interactions, time, cost, and belief support. Phased relaxation allows us to ignore problem features to get an initial heuristic, which we later adjust to bring back the ignored features. Planning graphs synergize with search by influencing pruning strategies (for example, helpful actions) and choosing objectives (as in partial satisfaction planning).

Planning graphs are versatile because of their many construction algorithms, the different information propagated on them, the types of problems they can solve, and the types of planners that employ them. We can construct serial, parallel, temporal, or labeled planning graphs. We can propagate level information, mutexes, costs, and labels. We can (as of now) solve classical, resource, temporal, conformant, conditional, and stochastic planning problems. They can be used in regression, progression, partial order, and GraphPlan-style planners. To this day we are finding new and interesting ways to use planning graphs.

As researchers continue to tackle more and more expressive planning models and problems, we have no doubt that reachability heuristics will continue to evolve to support search in those scenarios. In the near term, we expect to see increased attention to the development of reachability heuristics for hybrid planning models that simultaneously relax sets of classical planning assumptions, as well as models that support first-order (and lifted) action and goal languages. We hope that this survey will fuel this activity by unifying several views toward planning graph heuristics. As the scalability of modern planners improves through reachability heuristics, we are optimistic about the range of real-world problems that can be addressed through automated planning.

many discussions on planning graph heuristics. We would also like to thank Malte Helmert, Sungwook Yoon, Garrett Wolf, and the reviewers at *AI Magazine* for comments on earlier drafts of this article. This article is based in part on an invited talk at ICAPS 2003 titled "1001 Ways to Skin a Planning Graph for Heuristic Fun and Profit" and on a more recent tutorial given at ICAPS 2006.

Notes

1. Since this article is meant to be tutorial in nature, we have also prepared a set of slides to aid the presentation of this material. The slides are available at rakaposhi.eas.asu.edu/pg-tutorial.

2. In the following, we 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.

3. As we will see, the number of states can also be infinite (for example, when planning with resources).

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

5. 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, action 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).

6. Since action layers contain noop actions, technically speaking, mutexes can also exist between actions and propositions (through the associated noop actions), but mutexes are marked only between elements of the same layer.

7. With multiple objectives, it is necessary to find the Pareto set of nondominated plans.

8. We will see that multiple supporters are also needed for relaxed plans when there is uncertainty (see the "Nondeterministic Planning" section that follows).

9. Or, while we did not model this in our example, it is possible for the rover to warm up instruments used for sampling while driving to a rock.

References

Backstrom, C., and Nebel, B. 1995. Complexity Results for SAS+ Planning. *Computational Intelligence* 11(4): 625–655.

Benton, J.; Do, M. B., and Kambhampati, S. 2005. Over-Subscription Planning with Numeric Goals. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence,* 1207–1213. Denver, CO: Professional Book Center.

Blum, A., and Furst, M. L. 1995. Fast Planning through Planning Graph Analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence,* 1636–1642. San Francisco: Morgan Kaufmann Publishers.

Boddy, M.; Gohde, J.; Haigh, T.; and Harp, S. 2005. Course of Action Generation for Cyber Security Using Classical Planning. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS),* 12–21. Menlo Park, CA: AAAI Press.

Bonet, B., and Geffner, H. 1999. Planning as Heuristic Search: New Results. In *Proceedings of the 5th European Conference on Planning,* 360–372. Berlin: Springer-Verlag.

Bonet, B., and Geffner, H. 2000a. Planning as Heuristic Search. *Artificial Intelligence* 129(1–2): 5–33.

Bonet, B., and Geffner, H. 2000b. Planning with Incomplete Information as Heuristic Search in Belief Space. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems,* 52–61. Menlo Park, CA: AAAI Press.

Brafman, R., and Hoffmann, J. 2005. Contingent Planning Via Heuristic Forward Search with Implicit Belief States. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling,* 71–80. Menlo Park, CA: AAAI Press.

Bryce, D., and Kambhampati, S. 2004. Heuristic Guidance Measures for Conformant Planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling,* 365–375. Menlo Park, CA: AAAI Press.

Bryce, D., and Kambhampati, S. 2005. Cost Sensitive Reachability Heuristics for Handling State Uncertainty. In Proceedings of Twenty-First Conference on Uncertainty in Artificial Intelligence, 60–68. Eugene, OR: Association of Uncertainty in Artificial Intelligence.

Bryce, D., and Smith, D. 2006. Using Interaction to Compute Better Probability Estimates in Plan Graphs. Paper Presented at the Sixteenth International Conference on Automated Planning and Scheduling Workshop on Planning Under Uncertainty and Execution Control for Autonomous Systems, 6–10 June, The English Lake District, Cumbria, UK.

Bryce, D.; Kambhampati, S.; and Smith, D. 2006a. Planning Graph Heuristics for Belief Space Search. *Journal of Artificial Intelligence Research* 26: 35–99.

Bryce, D.; Kambhampati, S.; and Smith, D. 2006b. Sequential Monte Carlo in Probabilistic Planning Reachability Heuristics. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling,* 233–242. Menlo Park, CA: AAAI Press.

Cayrol, M.; Regnier, P.; and Vidal, V. 2000. New Results about LCGP, A Least Committed Graphplan. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems,* 273–282. Menlo Park, CA: AAAI Press.

Cushing, W., and Bryce, D. 2005. State Agnostic Planning Graphs and the Application to Belief-Space Planning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference,* 1131–1138. Menlo Park, CA: AAAI Press.

Do, M., and Kambhampati, S. 2003. Sapa: A Scalable Multi-Objective Metric Temporal Planner. *Journal of Artificial Intelligence Research* 20: 155–194.

Do, M., and Kambhampati, S. 2004. Partial Satisfaction (Over-Subscription) Planning as Heuristic Search. Paper presented at the Fifth International Conference on Knowledge Based Computer Systems, Hyderabad, India, 19–22 December.

Do, M. B.; Benton, J.; and Kambhampati, S. 2006. Planning with Goal Utility Dependencies. Paper Presented at the Sixteenth International Conference on Automated Planning and Scheduling Workshop on Preferences and Soft Constraints, The English Lake District, Cumbria, United Kingdom, 6–10 June.

Domshalk, C., and Hoffmann, J. 2006. Fast Probabilistic Planning through Weighted Model Counting. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling,* 243–251. Menlo Park, CA: AAAI Press.

Doucet, A.; de Freitas, N.; and Gordon, N. 2001. *Sequential Monte Carlo Methods in Practice.* New York: Springer.

Fikes, R., and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proceedings of Second International Joint Conference on Artificial Intelligence,* 608–620. Los Altos, CA: William Kaufmann, Inc.

Garrido, A.; Onaindia, E.; and Barber, F. 2001. A Temporal Planning System for Time-Optimal Planning. In *Proceedings of the Tenth Portuguese Conference on Artificial Intelligence,* 379–392. Berlin: Springer.

Gerevini, A.; Bonet, B.; and Givan, R. 2006. The Fifth International Planning Competition. The English Lakes District, Cumbria, United Kingdom.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through Stochastic Local Search and Temporal Action Graphs in LPG. *Journal of Artificial Intelligence Research* 20: 239–290.

Gerevini, A.; Saetti, A.; and Serina, I. 2005. Integrating Planning and Temporal Reasoning for Domains with Durations and Time Windows. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence,* 1226–1231. Denver, CO: Professional Book Center.

Ghallab, M., and Laruelle, H. 1994. Representation and Control in IXTET, A Temporal Planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, 61–67. Menlo Park, CA: AAAI Press.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. San Francisco: Morgan Kaufmann.

Haslum, P., and Geffner, H. 2000. Admissible Heuristics for Optimal Planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems,* 140–149. Menlo Park, CA: AAAI Press.

Helmert, M. 2004. A Planning Heuristic Based on Causal Graph Analysis. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, 161–170. Menlo Park, CA: AAAI Press.

Hoffmann, J., and Brafman, R. 2004. Conformant Planning Via Heuristic Forward Search: A New Approach. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling,* 355–364. Menlo Park, CA: AAAI Press.

Hoffmann, J., and Geffner, H. 2003. Branching Matters: Alternative Branching in Graphplan. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling,* 22–31. Menlo Park, CA: AAAI Press.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *Journal of Artificial Intelligence Research* 14: 253–302.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Journal of Artificial Intelligence Research* 20: 291–341.

Kambhampati, S., and Sanchez, R. 2000. Distance Based Goal Ordering Heuristics for Graphplan. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems,* 315–322. Menlo Park, CA: AAAI Press.

Kambhampati, S.; Lambrecht, E.; and Parker, E. 1997. Understanding and Extending Graphplan. In *Proceedings of Fourth European Conference on Planning*, 260–272. Berlin: Springer-Verlag.

Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. 1997. Extending Planning Graphs to an ADL Subset. In *Proceedings of Fourth European Conference on Planning*, 273–285. Berlin: Springer-Verlag.

Little, I.; Aberdeen, D.; and Thiebaux, S. 2005. Prottle: A Probabilistic Temporal Planner. In *Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference,* 1181–1186. Menlo Park, CA: AAAI Press.

Long, D., and Fox, M. 1999. Efficient Implementation of the Plan Graph in Stan. *Journal of Artificial Intelligence Research* 10: 87–115.

Long, D., and Fox, M. 2003. Exploiting a Graphplan Framework in Temporal Planning. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling,* 52–61. Menlo Park, CA: AAAI Press.

McDermott, D. V. 1996. A Heuristic Estimator for Means-Ends Analysis in Planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems,* 142–149. Menlo Park, CA: AAAI Press.

McDermott, D. 1998. PDDL—The Planning Domain Definition Language. Technical Report CVC TR-98-003 / DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT.

McDermott, D. 1999. Using Regression-Match Graphs to Control Search in Planning. *Artificial Intelligence* 109(1–2): 111–159.

Nebel, B. 2000. On the Compilability and Expressive Power of Propositional Planning Formalisms. *Journal of Artificial Intelligence Research* 12: 271–315.

Nguyen, X., and Kambhampati, S. 2000. Extracting Effective and Admissible State Space Heuristics from the Planning Graph. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence,* 798–805. Menlo Park, CA: AAAI Press.

Nguyen, X., and Kambhampati, S. 2001. Reviving Partial Order Planning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence,* 459–466. San Francisco: Morgan Kaufmann Publishers.

Nguyen, X.; Kambhampati, S.; and Nigenda, R. 2002. Planning Graph as the Basis for Deriving Heuristics for Plan Synthesis by State Space and CSP Search. *Artificial Intelligence* 135(1–2): 73–123.

Pednault, E. P. D. 1994. ADL and the State-Transition Model of Action. *Journal of Logic and Computation* 4(5): 467–512.

Refanidis, I., and Vlahavas, I. 2001. The GRT Planning System: Backward Heuristic Construction in Forward State-Space Planning. *Journal of Artificial Intelligence Research* 15: 115–161.

Rintanen, J. 2004. Distance Estimates for Planning in the Discrete Belief Space. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence,* 525–530. Menlo Park, CA: AAAI Press.

Ruml, W.; Do, M. B.; and Fromherz, M. P. J. 2005. On-Line Planning and Scheduling for High-Speed Manufacturing. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling,* 30–39. Menlo Park, CA: AAAI Press.

Sanchez, R., and Kambhampati, S. 2005. Planning Graph Heuristics for Selecting Objectives in Over-Subscription Planning Problems. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling* 192–201. Menlo Park, CA: AAAI Press.

Sanchez, J., and Mali, A. D. 2003. S-Mep: A Planner for Numeric Goals. In *Proceedings of the Fifteenth IEEE International Conference on Tools with Artificial Intelligence,* 274–283. Los Alamitos, CA: IEEE Computer Society.

Smith, D. E., and Weld, D. S. 1999. Temporal Planning with Mutual Exclusion Reasoning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence,* 326–337. San Francisco: Morgan Kaufmann.

Smith, D. E. 2003. The Mystery Talk. Invited Talk at the Second International Planning Summer School, 16–22 September.

Smith, D. E. 2004. Choosing Objectives in Over-Subscription Planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling,* 393–401. Menlo Park, CA: AAAI Press.

Van Den Briel, M.; Nigenda, R. S.; Do, M. B.; and Kambhampati, S. 2004. Effective Approaches for Partial Satisfaction (Over-Subscription) Planning. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence,* 562–569. Menlo Park, CA: AAAI Press.

Vidal, V. 2004. A Lookahead Strategy for Heuristic Search Planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling,* 150–160. Menlo Park, CA: AAAI Press.

Yoon, S.; Fern, A.; and Givan, R. 2006. Learning Heuristic Functions from Relaxed Plans. In *Proceedings of the Sixteenth International Conference on Auto-mated Planning and Scheduling,* 162–171. Menlo Park, CA: AAAI Press.

Younes, H., and Simmons, R. 2003. VHPOP: Versatile Heuristic Partial Order Planner. *Journal of Artificial Intelligence Research* 20: 405–430.

Zimmerman, T., and Kambhampati, S. 2005. Using Memory to Transform Search on the Planning Graph. *Journal of Artificial Intelligence Research* 23: 533–585.

Daniel Bryce is a Ph.D. candidate in the computer science department at Arizona State University, where he is a member of the Yochan research group. His current research interests include planning graph heuristics, planning and execution under uncertainty, and applications to computational systems biology. He earned a B.S. from Arizona State University in 2001 and anticipates earning his Ph.D. in 2007. His email address is dan.bryce@asu.edu.

Subbarao Kambhampati is a professor of computer science at Arizona State University. His group saw the connection between planning graph and reachability analysis in 1998, and subsequently developed several scalable planners including AltAlt, AltAlt-psp, SAPA, SAPA-ps, PEGG, CAltAlt, and POND. He is a 1994 NSF Young Investigator, a 2004 IBM faculty fellow, an AAAI 2005 program cochair, a JAIR associate editor, and a fellow of AAAI (elected for his contributions to automated planning. He received the 2002 college of engineering teaching excellence award. He gave an invited talk at ICAPS-03 entitled "1001 Ways to Skin a Planning Graph for Heuristic Fun and Profit." The current tutorial builds on that talk and discusses several new areas in which reachability heuristics have been found to be useful.