New Heuristics For Planning with Action Costs

Emil Keyder

TESI DOCTORAL UPF / 2010

Thesis advisor

Prof. Dr. Héctor Geffner,
Department of Information and Communication Technologies

UNIVERSITAT POMPEU FABRA

To my family

Acknowledgements

This thesis would not exist without the help and contributions of many people. First among these is my PhD advisor Hector Geffner, who went far beyond his responsibilities as an advisor in encouraging me and inspiring me to do this work. Hector is truly an exceptional researcher and person, and I count myself lucky for having spent these past five years as his student.

The stress of consecutive all-nighters trying to finish a paper before a conference deadline is much more bearable when shared with others in the same situation. Alexandre Albore, Nir Lipovetzky, Hector Palacios and Miquel Ramirez shared my frustrations and encouraged me and helped me with every aspect of my work, and I hope that I was able to do the same for them.

The planning community is a friendly one, and I am thankful for the discussions I had with Blai Bonet, Carmel Domshlak, Malte Helmert, Jörg Hoffmann, and Silvia Richter. These discussions expanded my horizons and deepened my insight into the problems I was working on. I am especially thankful to Blai Bonet for collaborating with me to prepare a tutorial for the International Conference on Planning and Scheduling, and Malte Helmert and Silvia Richter for their collaboration on the paper "Sound and Complete Landmarks for AND/OR Graphs". It was a pleasure to work with all of them. A special thanks goes to Blai Bonet, Malte Helmert, and Jörg Hoffmann for their generosity in making their software available and taking the time to reply to my sometimes foolish questions, as well as all the other researchers who allow their programs to be used. When I began this PhD, I had little idea of how to put together planners and other large pieces of software, and it is thanks to the community that I was able to learn these skills.

During my time in Barcelona, I had the extraordinary good luck of never having a bad flatmate. Alexandre Albore, Ayman Moghnieh, Filomena Nascimiento, Rita Rodrigues, and Xiaolei Yang all became dear friends, and I was very sad to see those who I no longer live with leave. My friends Candela Carrera, Sandra Gilabert, Ivan Herreros, Cenk Metin, Nardine Osman, Rida Sadek, and Nehir Sönmez all showed me new ways of looking at the world, and were there for me whenever I needed them.

I would like to especially thank Sira Ferradans for her love and understanding during the best part of these past five years; they would have been immeasurably poorer without her.

Finally, a big thanks to my family for their love and encouragement. It is not always easy to put up with me, and they have done an extraordinary job of it.

Abstract

Classical planning is the problem of finding a sequence of actions that take an agent from an initial state to a desired goal situation, assuming deterministic outcomes for actions and perfect information. Satisficing planning seeks to quickly find low-cost solutions with no guarantees of optimality. The most effective approach for satisficing planning has proved to be heuristic search using non-admissible heuristics. In this thesis, we introduce several such heuristics that are able to take into account costs on actions, and therefore try to minimize the more general metric of cost, rather than length, of plans, and investigate their properties and performance. In addition, we show how the problem of planning with soft goals can be compiled into a classical planning problem with costs, a setting in which cost-sensitive heuristics such as those presented here are essential.

Resumen

La planificación clásica es el problema que consiste en hallar una secuencia de acciones que lleven a un agente desde un estado inicial a un objetivo, asumiendo resultados determinísticos e información completa. La planificación "satisficing" busca encontrar una solución de bajo coste, sin garantías de optimalidad.

La búsqueda heurística guiada por heurísticas no admisibles es el enfoque que ha tenido mas éxito. Esta tesis presenta varias heurísticas de ese género que consideran costes en las acciones, y por lo tanto encuentran soluciones que minimizan el coste, en lugar de la longitud del plan. Además, demostramos que el problema de planificación con "soft goals", u objetivos opcionales, se puede reducir a un problema de planificación clasica con costes en las acciones, escenario en el que heurísticas sensibles a costes, tal como las aquí presentadas, son esenciales.

Preface

Classical planning is the problem of finding a sequence of operators, or plan, that given an initial state achieves a desired goal situation. It can be seen as a pathfinding problem in an implicitly defined graph whose nodes represent states, specified in terms of the values of a set of variables, and whose edges represent operators, specified in terms of sets of requirements and effects on the values of those variables. As in the pathfinding problem, each edge or operator in this graph is associated with a weight or cost, and among solution paths, those with lower weight are preferred. Planning approaches can be divided into two categories based on the solutions they seek: optimal planning, in which plans are guaranteed to have minimum cost, and satisficing planning, which finds some plan without offering any guarantee of its optimality.

Heuristic search with a heuristic function that given a state estimates the cost of reaching a goal state has proven to be the most effective technique in both of these settings. First introduced by Bonet et al. (1997) and McDermott (1996), it greatly increased the size and complexity of problems that domain-independent planners were able to solve. Current state-of-the-art approaches continue to employ this approach in both the satisficing (Richter and Westphal, 2010) and optimal (Helmert and Domshlak, 2009) settings. Improvements in the performance of heuristic search arise from two factors: improvements in search algorithms, including the way in which heuristic estimates are used, and improvements in heuristic functions resulting in more accurate estimations of the cost of states. This thesis focuses mainly on improving the accuracy of heuristic functions, with a special emphasis on problems in which the costs of actions are not uniform.

Heuristic functions generally result from solving a relaxation or simplification of a given problem. Part II of this thesis focuses on a relaxation of the planning problem that has been widely used for this purpose, the delete relaxation. The cost of an optimal solution to the delete-relaxation is a very informative lower bound on the cost of a planning problem, yet such solutions cannot be found in polynomial time. However, various approximations that approach its cost from both above and below have been proposed, giving both admissible (non-overestimating) heuristics for use in optimal planning and non-admissible heuristics for use in satisficing planning. In this thesis, we consider two lines of research in non-admissible delete-relaxation heuristics and show how they can be unified to obtain heuristics that combine the strengths of each, such as taking into account action costs and producing explicit solutions to the delete relaxation problem, which can be exploited beyond what a simple cost estimate allows. We demonstrate that one of these lines of research implicitly makes use of a simplification known as the *independence assumption* that is explicit in the other, and relate this property to research into several graph problems which turn out to be equivalent to the delete relaxation. In addition, we propose a novel non-admissible heuristic that approximates the cost of this problem without relying on the independence assumption, and show that it greatly improves the informativeness of the resulting estimates in certain problems.

In Part III, we describe two techniques for deriving heuristic estimates that take into account delete information in a problem. The first of these consists of a method for finding *landmarks*, values that a variable must take on at some point in the execution of every valid plan, or actions that must occur in every plan. We state the equations giving the complete set of delete relaxation landmarks, which can be solved in polynomial time, and show how to apply our techniques to a problem resulting from a previously introduced translation of a planning problem, allowing for the first time the discovery of landmarks that go beyond the delete relaxation. Such landmarks can then be used in previously introduced cost-partitioning heuristics that split the costs of operators among the various landmarks that they achieve. The second technique explores the idea of a new kind of invariant in planning, that of *choice variables*. Choice variables are multivalued variables to which each plan can assign a value at most once, and are similar in spirit to the variables used in problems such as constraint satisfaction and graphical models. We show that a heuristic that reasons about different possible assignments to these variables is more informative in certain settings, and that its values can be computed efficiently by adapting pre-existing methods from the field of graphical models to planning. Furthermore, we give planning encodings of some well-known problems from various fields of computer science for which the heuristic is optimal.

Part IV presents a compilation of a non-classical planning problem, that of planning with soft goals, into a classical planning problem with action costs. Soft goals are goals that are not necessarily achieved by every valid plan but are associated with some reward or utility. We show that problems with soft goals can be transformed into classical planning problems in which the penalty of not obtaining a reward must be traded off against the cost of the plan that achieves a soft goal. The cost-sensitive heuristics described in the previous sections are especially useful in solving such compilations.

Some of the work presented in this thesis has previously been published in the following articles:

- Emil Keyder, Silvia Richter, and Malte Helmert. *Sound and Complete Landmarks for And/Or Graphs*. In *Proceedings of the 19th European Conference on Artificial Intelligence*, pages 335–340, 2010. [Chapter 5]

- Emil Keyder and Hector Geffner. *Soft Goals Can Be Compiled Away*. In *Journal of Artificial Intelligence Research*, Volume 36, pages 547–556, 2009. [Chapter 7]

- Emil Keyder and Hector Geffner. *Trees of Shortest Paths vs. Steiner Trees: Understanding and Improving Delete Relaxation Heuristics*. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1734–1739, 2009. [Chapter 4]

- Emil Keyder and Hector Geffner. *Heuristics for Planning with Action Costs Revisited.* In *Proceedings of the 17th European Conference on Artificial Intelligence*, pages 588–592, 2008. [Chapter 3]

- Emil Keyder and Hector Geffner. *Set-Additive and TSP Heuristics for Planning with Action Costs and Soft Goals.* In *Proceedings of the ICAPS-07 Workshop on Heuristics for Domain-Independent Planning*, 2007. [Chapter 3]

- Emil Keyder and Hector Geffner. *Heuristics for Planning with Action Costs.* In *Proceedings of the 12th Spanish Conference on Artificial Intelligence*, pages 140–149, 2007. [Chapter 3]

Contents

List of Figures

List of Tables

Part I

Background

CHAPTER 1

The Planning Problem

In this chapter we present the classical planning model, and discuss how it can be described implicitly by factored representations. We then briefly review some complexity results for the planning problem in terms of the size of these representations, and discuss the heuristic search approach for solving planning problems.

1.1 The Classical Planning Model

Classical planning is the problem of finding a sequence of actions, or *plan*, that when applied in the initial state of the problem results in a goal state. A classical planning problem can be viewed as a directed graph whose nodes represent states of the planning problem, and whose edges represent actions that change the state from that represented by the source node of the edge to that represented by the target node. A plan is then a path from the node in the graph representing the initial state to a node representing a state that is an element of the set of goal states of the problem. The formal model underlying the planning problem can be described as follows:

Definition 1.1 (Classical planning model)**.** *A planning model* $\Pi = \langle S, s_0, S_G, O, a, f\rangle$ *consists of:*

- *A finite and discrete set of states S,*
- *An initial state $s_0 \in S$,*
- *A set of goal states $S_G \subseteq S$,*
- *A set of operators O,*
- *The* applicability *function $a : S \mapsto \mathcal{P}(O)$ giving the set of operators that are applicable in each state,*
- *The* transition *function $t : S \times O \mapsto S$, where $t(s, o)$ is defined when $o \in a(s)$.*

We say that the state resulting from applying an operator o in a given state s is $t(s, o)$, and denote this resulting state as $s[o]$. The result of applying a sequence of operators $\langle o_1, \ldots, o_n \rangle$ to a state can be defined recursively as

$$\begin{aligned} s[\epsilon] &= s \\ s[o_1, \ldots, o_n] &= (s[o_1, \ldots, o_{n-1}])[o_n] \end{aligned}$$

Definition 1.2 (Plan)**.** *Given a planning model Π, a* plan *π for Π is a sequence of operators such that $s_0[\pi] \in S_G$. The length $|\pi|$ of a plan is the number of operators in a plan.*

In the absence of a cost function, plans with lower length are preferred to those with higher length. Alternatively, the planning model may be stated in conjunction with a *cost function* that assigns costs to the operators of the problem:

Definition 1.3 (Classical planning model with costs)**.** *A planning model with costs Π_c consists of a planning model $\Pi = \langle S, s_0, s_G, O, a, t \rangle$ along with a function $c : O \mapsto \mathbb{R}_0^+$ that maps each operator in the model to a non-negative cost.*

Definition 1.4 (Plan cost)**.** *The cost of a plan $\pi = \langle o_1, \ldots, o_n \rangle$ is given by*

$$cost(\pi) = \sum_{i=1}^{n} c(o_i)$$

In planning problems with costs, plans with lower cost are preferred to plans with higher cost. Planning problems without cost, in which the length $|\pi|$ is the criteria for preference, can be seen as a special case of planning with costs in which $c(o) = 1$ for all $o \in O$.

1.2 Factored Representations in Planning

Since it is impractical to explicitly enumerate the state spaces of larger planning problems, *factored* representations, in which states are represented by complete assignments to a set of variables with finite and discrete domains, are commonly used. Given such a representation, the sets of goal states and operators, as well as the applicability and transition functions, can also be described in terms of these variables.

The most common and simplest factored representation is one which consists of only boolean variables, with the value of each such variable indicating whether a proposition about the world is true or false in a given state. Such variables are also known as *fluents* or *facts*. This representation was first used in the Stanford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson, 1971) and is known by that name today:

Definition 1.5 (STRIPS). *A planning problem in* STRIPS, $\Pi = \langle F, O, I, G \rangle$ *consists of:*

- *A set of boolean variables (fluents) F,*
- *A set of tuples O representing operators, each having the form $\langle Pre(o), Add(o), Del(o) \rangle$, where $Pre(o)$, $Add(o)$, $Del(o)$ $\subseteq F$*
- *A set $I \subseteq F$, describing the initial state*
- *A set $G \subseteq F$, describing the set of goal states.*

Note that in this representation, preconditions of operators are assumed to be positive. If operators having preconditions that require some fluent p to be false in the current state are desired, this can easily be achieved by introducing a second fluent p' that has the value true whenever p false, and vice versa (Bylander, 1994). All operators adding p must then additionally delete p', and all operators deleting p must add p'.

Definition 1.6 (STRIPS with costs). *A* STRIPS *problem with costs* $\Pi = \langle F, O, I, G, c\rangle$ *consists of:*

- *The* STRIPS *problem defined by* $\Pi = \langle F, O, I, G\rangle$
- *A cost function* $c : O \mapsto \mathbb{R}_0^+$ *that assigns to each* STRIPS *operator in the problem a non-negative cost.*

The STRIPS representation describes the state space graph underlying the planning problem implicitly. Each subset $s \subseteq F$ of the set of fluents represents a state in which the fluents $p \in s$ have the value *true*, while the fluents $p' \in F \setminus s$ have the value *false*. I then represents the unique initial state s_0, while G represents the set of goal states, defined as $S_G = \{s \mid G \subseteq s\}$. The applicability and transition functions a and t are obtained from the set of *ungrounded* operators O, each of which may correspond to more than one operator in the *grounded* (explicit) representation of the problem: $a(s) = \{o \mid \mathit{Pre}(o) \subseteq s\}$, and $t(s, o) = (s \cup \mathit{Add}(o)) \setminus \mathit{Del}(o)$.

More complicated factored representations such as SAS$^+$ (Bäckström and Nebel, 1995) describe planning problems in terms of variables with domains of arbitrary (finite) size. Like STRIPS, these representations also allow for the underlying state space to easily be made explicit. Such representations are not used in the work presented in this thesis, and we do not discuss them here.

1.3 Complexity

Given the explicit representation of a planning problem Π, the problem of finding the optimal paths from the single source s_0 to all $s \in S$ (and therefore all $s \in S_G$) is in P, and can be solved by Dijkstra's algorithm in time $O(|S|^2)$ (Cormen et al., 2001).

However, the state spaces of planning problems are typically too large even for the purposes of enumeration. The complexity of the planning problem is therefore analyzed in terms of the size of the factored representation used.

Definition 1.7 (Plan Existence)**.** *Given a factored representation of a planning problem* Π*,* ***PlanExt***(Π) *is the following decision problem:*

`INSTANCE:` *A planning problem* Π*.*

`QUESTION:` *Does a plan* π *for* Π *exist?*

Definition 1.8 (Plan Cost)**.** *Given a factored representation of a planning problem* Π *and a constant value* $k \in \mathbb{R}_0^+$*,* ***PlanCost***(Π) *is the following decision problem:*

`INSTANCE:` *A planning problem* Π*, a constant* $k \in \mathbb{R}_0^+$*.*

`QUESTION:` *Does a plan* π *for* Π *with cost*$(\pi) \leq k$ *exist?*

For planning problems expressed in STRIPS, both `PlanExt`(Π) and `PlanCost`(Π, k) are PSPACE-complete (Bylander, 1994). Given the theoretical difficulty of the planning problem, approaches to planning are generally evaluated in terms of their practical performance, rather than worst-case complexity guarantees.

1.4 Planning as Heuristic Search

Given the interpretation of planning problems as directed graphs whose nodes represent states and whose edges represent actions, graph search algorithms are a logical choice for solving them. Yet due to the size of the state spaces of planning problems, "blind" approaches such as Dijkstra's algorithm are in general impractical. However, *heuristic* search approaches using a heuristic function that is computed automatically given the problem representation have proven to be tremendously successful in recent years, with winners of several of the most recent International Planning Competitions (IPC), including the most recent one, using this strategy (Richter and Westphal, 2010; Hoffmann and

Nebel, 2001; Helmert, 2006; Bonet and Geffner, 2001). The primary focus of this thesis is on how to improve the quality and informativeness of such automatically extracted heuristics. In this section, we briefly review heuristics and search in general and discuss some specific search algorithms.

Heuristics

A *heuristic* is a way of choosing between different courses of action in order to achieve some goal (Pearl, 1983). Heuristics make no guarantees as to the optimality of their suggestions, yet in combination with heuristic search algorithms have been used successfully to find both optimal and satisficing solutions to many problems in which the underlying search space is too large to find a solution using exhaustive search methods. Canonical examples are problems such as the 15-puzzle, in which tiles must be moved around a grid to reach a target configuration, and the route finding problem, in which a route must be found between two cities on a highway map. In many settings, heuristics take the form of *heuristic estimators* that compute for a given state an estimate of the cost of reaching the goal from that state:

Definition 1.9 (Heuristic estimator)**.** *A heuristic estimator is a function $h : S \mapsto \mathbb{R}_0^+$ that given a state s estimates the cost of reaching a goal state from s.*

One suggested course of action a in the state s is then the one which minimizes the estimated cost $h(s')$ of the state $s' = t(s, a)$ which results from taking it. While such a state represents the quickest way of getting to the goal from the current situation, if different courses of action have different costs, these must also be considered if a globally optimal solution is desired.

While its computation in practice for problems that are of interest to heuristic search is impractical, it is useful to define the *perfect heuristic estimator* h^*, which at any state s has the value of the cost of an optimal solution from that state:

Definition 1.10 (h^*)**.** *The perfect heuristic estimator $h^*(s)$ is the cost of an optimal (lowest-cost) solution to a problem from state s.*

The fact that h^* gives the optimal cost from the current state s means that there should always exist an action a such that $h^*(s) = cost(a) + h^*(t(s,a))$. This in turn implies that it should be possible to go from the initial state of the problem to a goal state by following a sequence of states for which the sum of the accumulated cost until that state and the h^* value of the state are constant. In practice, heuristics for problems of interest are far less accurate, and require the exploration of a much larger part of the state space.

Admissible, or non-overestimating heuristic estimators can be defined with reference to h^*:

Definition 1.11 (Admissible heuristic estimator)**.** *(Pearl, 1983) An admissible heuristic estimator h is one which satisfies $h(s) \leq h^*(s)$ for all states $s \in S$.*

An admissible heuristic estimator for the route finding problem, for example, is the Euclidean distance in the map between two cities, as this is guaranteed to be a lower bound on the actual minimum distance that must be driven to get from one to the other. Admissible heuristic estimators are important for problem solving, as they can be used in combination with search algorithms that always explore first courses of action that appear to be less costly in order to obtain optimal solutions. In contrast, non-admissible heuristic estimators may lead to suboptimal solutions, as an overestimate of the optimal cost of a state may cause the algorithm to rule out a solution that actually has lower cost. Typically, both types of heuristic estimators are defined as the costs of solutions to simpler versions of the problem than the one at hand. These simpler problems are called *relaxations* of the original problem, and result from ignoring certain of its properties or complexities (Pearl, 1983). When any solution to a problem also constitutes a solution to a specific relaxation of the problem, the relaxation has the property that the cost of its optimal solutions constitute an admissible heuristic estimator, since the cost of the optimal solution to the original problem is an upper bound on the cost of the relaxation. Part II of this thesis is dedicated to improving solutions to the delete relaxation in planning, which has this property and results from ignoring the delete effects in a problem specified in STRIPS.

Search

A *heuristic search algorithm* is a procedure that uses a heuristic estimator to find a sequence of actions that reach a goal. The state space of the problem being searched is represented by the algorithm in the form of *nodes*, which are structures which contain the description of the state they represent along with some further information, such as the action that was used to reach that state and the total cost of the actions used in the path currently leading to that state. Storing the action that led to each state allows the algorithm to recover a path from the initial state of the problem to a state represented by any node under consideration by following the actions backwards from the current node until the node representing the initial state is encountered.

Best first search (BFS) algorithms explore the search space by ranking all of the currently reachable nodes whose successors have not yet been generated according to some evaluation function $f(n)$, which is typically a linear combination of the accumulated cost of reaching the node, denoted by $g(n)$, and the estimated cost of reaching the goal from that node according to the heuristic estimator used, written as before $h(n)$ (Pearl, 1983). The algorithm operates by placing the unexpanded nodes, which are those whose successors have not yet been generated, in a set in which each is associated with its value according to the ranking function $f(n)$, known as an *open list* (Algorithm 1.1). At each step of the algorithm, a node n with minimum $f(n)$ is removed from the open list, and its successors are inserted with the $f(n)$ value that results from their evaluation. In order to avoid re-exploring previously considered states, those nodes that have been removed from the open list and expanded are placed in a set of nodes known as the *closed list*, and when a node is generated the closed list and open lists are checked to see if it had already been generated previously. If it was already present in the closed list or open list with a worse $f(s)$, this value and the action stored as leading to the node are updated, and if it was found on the closed list, the state is moved back to the open list.

A generic form of the typical evaluation function used by heuristic search algorithms is given by $f(n) = g(n) + w(h(n))$. When $w = 1$, this results in the A* algorithm, which finds optimal

```
Input: A problem model of the form
       $\Pi = \langle S, s_0, S_G, O, a, f \rangle$
Output: A path to a goal state $s_g$

open-list ← a set of nodes
$n_i$ ← a node representing the initial state
Calculate $f(n_i)$ and add $n_i$ to open-list
while open-list $\neq \emptyset$ do
    $n$ ← a node with minimal $f(n)$ in open-list
    Remove $n$ from open-list
    if IsGoal($n$) then
        return GeneratePath($n$)
    for $n' \in$ GenerateSuccessors($n$) do
        Calculate $f(n')$
        if $n' \notin$ open-list, closed-list then
            Add $n'$ to open-list
        else if previous $f(n')$ > current $f(n')$ then
            Update the path that lead to $n'$ and its $f(n')$
            value
            if $n' \in$ closed-list then
                Move $n'$ to open-list
    Place $n$ in closed-list
```

Figure 1.1: The best-first search algorithm.

solutions when used in combination with admissible heuristics (Pearl, 1983). For higher values of w, the value of the heuristic estimator for a state is given more weight, and states estimated to be closer to a goal state, even if they have higher g values, are expanded more aggressively. When the value of g is ignored completely by the evaluation function and $f(s) = h(s)$, the search takes into account only the estimated distance to the goal, ignoring the cost accumulated in the solution path to get to that state. This type of search is known as *greedy best-first search* (GBFS), and is the search algorithm that we use for the experimental results presented in chapters 3 and 4.

A number of variations and improvements have been proposed to the generic BFS algorithm in the context of planning. We now briefly review those that are used in the experimental results presented in this thesis.

Helpful Actions. The vast majority of the computational effort expended by heuristic search planners is in the calculation of the heuristic estimator $h(s)$. Techniques that allow the planner to avoid generating all of the possible successors of a state when it is expanded therefore allow such planners to scale up to much larger problems. One such technique is that of *helpful actions* (Hoffmann and Nebel, 2001; Helmert, 2006) [1]. Heuristics with helpful actions, in addition to estimating the cost to reach the goal from a given state, also return a subset of the applicable actions in that state that they consider particularly promising:

Definition 1.12 (Heuristic estimator with helpful actions)**.** *A* heuristic estimator with helpful actions *is a function* $h : S \mapsto \mathbb{R}_0^+ \times \mathcal{P}(a(s))$*, where* $a(s)$ *is the set of applicable operators in the evaluated state* s*.*

In heuristics that are based on relaxations, such actions are typically those used in the solution to the relaxation found by the heuristic that are applicable in the current state. Various search frameworks have been proposed to take advantage of helpful actions, in which either states resulting from non-helpful actions are not generated, or states resulting from helpful actions are expanded preferentially. The approach used by the FF planner was to initially attempt to find a solution while considering only states resulting from helpful actions, and resorting to normal exploration if this failed. A more recently proposed approach used by several successful planners is to alternate between exploring states resulting from helpful actions and those resulting from all other actions (Helmert, 2006). This leads to a search algorithm that is able to take advantage of the extra information presented by the heuristic in the form of helpful actions without sacrificing completeness. Algorithmically, it can be implemented in the form of a best first search with *multiple open lists*.

[1] The term "helpful actions" was first used in the context of the FF planner (Hoffmann and Nebel, 2001). A similar concept called "preferred operators" was discussed by Helmert (2006). Here we refer to the general approach as "helpful actions".

Multiple open lists. Multiple open lists can be seen both as a tool for integrating multiple heuristic estimators (Richter and Westphal, 2010) into a single search algorithm and for preferentially evaluating nodes resulting from the application of helpful actions without sacrificing completeness (Helmert, 2006). Here we focus on their interaction with helpful actions. A BFS search using a heuristic with helpful actions h with two open lists works as described above, except that when it generates the successors to a node, it places all nodes resulting from the application of helpful actions on one of the open lists, and those resulting from non-helpful actions on the other. When choosing a state to expand, the algorithm then alternates between the two open lists in choosing nodes of minimum $f(n)$. As the number of actions chosen as helpful is typically small compared to the number of all applicable actions in a state, this results in a much deeper exploration of the state space using only helpful actions. Yet since all nodes are eventually evaluated, this does not affect the completeness of the search even in the case in which the helpful actions chosen by the heuristic are not of high quality. To further promote a fast exploration of the state space, the proportions with which two open lists are used to select a node to expand can be changed, for example removing a node from the open list containing only nodes resulting from helpful actions p times for every node that is removed from the other open list (Richter and Westphal, 2010). In the experiments described in this paper, we do not use this technique, instead choosing nodes from both open lists with equal frequency.

Delayed Evaluation. As stated above, the computation of heuristic estimators' values is typically the largest computational bottleneck faced by heuristic search planners. Furthermore, in many cases a large proportion of the nodes that are placed in the open list are never expanded, leading to many heuristic evaluations that serve only to disqualify a node from immediate consideration. *Delayed evaluation* seeks to minimize the impact of this situation by not evaluating the heuristic estimator for a node until it is expanded, using the heuristic estimate computed for its parent node in the calculation of the evaluation function $f(n)$ (Helmert, 2006; Richter and Westphal, 2010). While this results in a less informative evaluation function for individual nodes, this is offset by the search algorithm's being able to quickly generate a larger portion of the state space.

1.5 Summary

The classical planning model is that of a state space with a single initial state and a set of goal states. In factored representations, the states and edges of the state space are represented implicitly: the states in terms of a complete assignment to a set of variables, and the edges between them in terms of operators with preconditions and effects that are partial assignments to this set of variables. The planning problem is then PSPACE-complete in terms of the size of this representation. The heuristic search approach to planning consists of the use of a heuristic extracted automatically from the factored problem representation in combination with standard heuristic search methods to find a path from the initial state of the state space to one goal state. Parts II and III of this thesis describe a number of ways in which such heuristics can be defined and computed.

Part II

Delete Relaxation Heuristics

Chapter 2

The Delete Relaxation

In this chapter we discuss the intuition behind the delete relaxation and how it is constructed in the STRIPS formalism. We show how it can be seen as an instance of a pathfinding problem in two distinct formulations of graphs that are extensions of standard directed graphs, and isolate the sources of complexity of the problem.

2.1 Introduction

Heuristics are commonly derived from relaxations of problems in which certain conditions are removed, or certain assumptions are made (Pearl, 1983). Planning is no exception, and one of the most fruitful tools for the derivation of new heuristics has been the delete relaxation, with the heuristics used in successful planners such as HSP (Bonet and Geffner, 2001), FF (Hoffmann and Nebel, 2001), and LAMA (Richter and Westphal, 2010) based on solutions to it. Simply put, the delete relaxation considers a factored representation of a planning problem and makes the assumption that once a variable is assigned a certain value during the execution of a plan, it continues to hold this value simultaneously with its previously held value(s). It follows from this that a variable may have several values at once, as different actions in the plan may assign new values to the variable.

Given a problem, the cost of the optimal solution to the associated delete-relaxation problem is a lower bound on its cost and can therefore be used as an admissible heuristic. Intuitively, this follows from the fact that any plan for the original problem is also a plan for the delete relaxation, and since a plan in the original problem may have to assign a variable a certain value multiple times, cheaper plans may exist in the delete relaxation as a variable need be given a certain value at most once.

2.2 The Delete Relaxation in STRIPS

The delete relaxation problem was first formulated in terms of the STRIPS representation, and was used to derive the heuristic used in the seminal heuristic search planner HSP (Bonet and Geffner, 2001). Given a problem Π expressed in STRIPS, a simple transformation consisting of removing all delete effects from operators results in a second STRIPS problem Π^+ which encodes its delete relaxation:

Definition 2.1 (Delete relaxation in STRIPS)**.** *Given a* STRIPS *problem* $\Pi = \langle F, O, I, G\rangle$*, its* delete relaxation Π^+ *is described by the tuple* $\Pi^+ = \langle F, O^+, I, G\rangle$*, where*

$$O^+ = \{\langle \mathit{Pre}(o), \mathit{Add}(o), \emptyset\rangle \mid o \in O\}$$

The delete relaxation of a STRIPS problem with costs is obtained in the same way, with no change to the cost function c.

Definition 2.2 (Relaxed plan)**.** *Given a* STRIPS *problem* $\Pi = \langle F, O, I, G\rangle$*, a relaxed plan for* Π *is a plan for its delete relaxation* Π^+*.*

For a planning problem Π^+ with no deletes, the plan existence problem `PlanExt`(Π^+) is in P(Bylander, 1994). This property is due to the fact that the set of reachable fluents in the problem grows monotonically with the application of operators, and operators whose preconditions are newly satisfied can be applied until no further fluents can be achieved. Since no operator need be applied more than once, this occurs after at most $|O|$ iterations. If the goals of the problem are then contained in the set of reachable fluents, a plan exists. The problem of finding an optimal plan, however, or more generally the $\texttt{PlanCost}(\Pi^+, k)$ problem, is NP-complete even if operators are restricted to have no preconditions (Bylander, 1994). This follows from the fact that the vertex cover problem, for instance, can be easily reduced to $\texttt{PlanCost}(\Pi^+, k)$, with operators representing vertices and having the add effects of edges which are incident to them, and the goals of the problem being the full set of edges of the graph. Planning heuristics for satisficing planning therefore commonly attempt to compute a good approximation to the cost of the optimal plan for Π^+ in polynomial time, while providing no guarantees of optimality, and as follows logically no guarantees of admissibility.

2.3 Directed Hypergraphs and AND/OR graphs

The delete relaxation problem can be seen as a pathfinding problem on an instance of one of two types of graph: a *directed hypergraph* or an AND/OR *graph*. We note that due to this equivalence, all of the methods proposed in this thesis for better approximating the optimal cost of the delete relaxation can also be seen as methods for finding paths in either graph representation.

Directed Hypergraphs

Directed hypergraphs are a generalization of directed graphs and hypergraphs, in which each edge may have an arbitrary number of source and target vertices (Ausiello et al., 1998; Gallo et al., 1992) [1]:

Definition 2.3 (Directed hypergraph)**.** *A directed hypergraph $\mathcal{H} = \langle V, E \rangle$ consists of a set of vertices V and a set of edges E,*

[1] Alternatively, only multiple source vertices may be allowed. This is not an essential distinction as allowing multiple target vertices does not increase the complexity of the problem, see Proposition 2.10 for details.

where each edge $e_i = \langle s(e_i), t(e_i)\rangle$ is defined by a set of source nodes $s(e_i) \subseteq V$ *and a set of target nodes* $t(e_i) \subseteq V$*.*

Given a hypergraph $\mathcal{H}$, the *subhypergraph* induced by a set of edges $E' \subseteq E$ is the hypergraph $\mathcal{H}_{E'} = \langle \bigcup_{e_i \in E'} s(e_i) \cup t(e_i), E'\rangle$. The criteria for a subgraph induced by a set of edges E' constituting a *hyperpath* between a set of nodes I and a set of nodes G in a directed hypergraph echoes the intuition behind the notion of paths in standard directed graphs:

Definition 2.4 (Directed hyperpath)**.** *Given a hypergraph $\mathcal{H} = \langle V, E\rangle$, a* directed hypergraph *from $I \subseteq V$ to $G \subseteq V$ is a subhypergraph $\mathcal{H}_{E'}$ such that for all $g \in G$, either*

- *$g \in I$, or*
- *there exists an edge $e_i \in E'$ such that $g \in t(e_i)$, and $\mathcal{H}_{E'}$ constitutes a path from I to $s(e_i)$.*

Note that when the graph is a standard directed graph, i.e. $|s(e_i)| = |t(e_i)| = 1$ for all $e_i \in E$, this definition is equivalent to the standard definition of a path in a directed graph.

A weighted directed hypergraph is a triple $\mathcal{H} = \langle V, E, w\rangle$, where $w : E \mapsto \mathbb{R}_0^+$ is a function that assigns a non-negative weight to each edge in the graph. The weight of a (sub)hypergraph is $c(\mathcal{H}) = \sum_{e \in E} w(e)$.

The problem of solving the delete relaxation optimally can be stated as a pathfinding problem on a hypergraph that can be constructed from the planning problem. Given a planning problem $\Pi = \langle F, O, I, G, c\rangle$ with no deletes, we define the directed hypergraph $\mathcal{H}^\Pi$ corresponding to Π as $\mathcal{H}^\Pi = \langle F, \{\langle Pre(o), Add(o)\rangle \mid o \in O\}, c\rangle$.

Proposition 2.5 (Equivalence of directed hypergraphs)**.** *Let π constitute a plan for a* STRIPS *problem* Π *with no deletes. Then $\mathcal{H}^\Pi_{E'}$, where $E' = \{\langle Pre(o), Add(o)\rangle \mid o \in \pi\}$ constitutes a path in $\mathcal{H}^\Pi$ from I to G, and $w(\mathcal{H}^\Pi_{E'}) = cost(\pi)$.*

Various heuristics that are in common use for the delete relaxation have been proposed in the context of directed hypergraphs as measures of cost for which the problem of finding optimal hyperpaths is tractable. In particular, the *traversal cost* and *rank* of hyperpaths are measures for which minimum weight hyperpaths can be found in polynomial time (Ausiello et al., 1998), and correspond respectively to the additive heuristic h^{add} and the max heuristic h^{max} (Bonet and Geffner, 2001) which will be discussed in Chapter 3. Additionally, the *unfolded hyperpath* structure used to define these measures turns out to closely mirror the idea behind the independence assumption upon which these heuristics are based, and gives new insight into other heuristics employing it.

AND/OR Graphs

AND/OR graphs are another generalization of directed graphs in which the set of nodes is partitioned into two disjoint sets, the AND nodes V_{and}, and the OR nodes V_{or}. The role taken by edges with multiple source nodes in hypergraphs is taken here by AND nodes. Directed graphs can be seen as a special case of AND/OR graphs in which all nodes are OR nodes.

Definition 2.6 (AND/OR graph)**.** *An* AND/OR graph *is a directed graph $G = \langle V, E\rangle$ in which the set of nodes V is partitioned into two disjoint sets V_{and}, V_{or}.*

Solutions in AND/OR graphs take the form of *justifications* for a set of initial nodes $V_{\text{I}} \subseteq V$ and a set of goal nodes $V_{\text{G}} \subseteq V$:

Definition 2.7 (AND/OR graph justification)**.** *A* justification *in an* AND/OR *graph $G = \langle V, E\rangle$ for a set of initial nodes $V_{\text{I}} \subseteq V$ and a set of goal nodes $V_{\text{G}} \subseteq V$ is a subgraph $J = \langle V^J, E^J\rangle$ of G such that:*

- $V_{\text{G}} \subseteq V^J$
- $\forall a \in (V^J \cap V_{\text{and}}) \setminus V_{\text{I}} : \forall \langle v, a\rangle \in E : v \in V^J \wedge \langle v, a\rangle \in E^J$
- $\forall o \in (V^J \cap V_{\text{or}}) \setminus V_{\text{I}} : \exists \langle v, o\rangle \in E : v \in V^J \wedge \langle v, o\rangle \in E^J$

- *J is acyclic*

The intuition behind a justification in an AND/OR graph is that all predecessors of an AND node $v \in V_{\text{and}}$ must be proved to be true, or reached, in order to prove that node true, while to prove or reach an OR node $v \in V_{\text{or}}$, it is sufficient to prove any one of its predecessors true. Given an AND/OR graph and a set of initial nodes V_{I}, the set of initial nodes are nodes that are assumed to be true, and correspond to the facts that are true in the initial state of a planning problem. Given a weight function $w : V \mapsto \mathbb{R}_0^+$, the weight of a justification is defined as $w(J) = \sum_{v \in V^J} w(v)$.

Finding a plan for a problem Π with no deletes is equivalent to finding a minimum-weight justification in an AND/OR graph G^{Π} that can be constructed from a planning problem Π^+ with no deletes. The set of AND nodes V_{and} for this graph is given by the set of operators O of Π^+, with each $v \in V_{\text{and}}$ having weight $w(v)$ equal to the cost of the corresponding operator, the set of OR nodes V_{or} by the set of fluents F, with each having weight 0, and the set of edges E by $E = \bigcup_{o \in O} E^{pre(o)} \cup E^{add(o)}$, where

- $E^{pre(o)} = \{\langle f, o \rangle \mid f \in Pre(o)\}$
- $E^{add(o)} = \{\langle o, f \rangle \mid f \in Add(o)\}$

Proposition 2.8. *Given a planning problem Π with no deletes, the sets of AND nodes $V_{\text{and}} \cap V^J$ of minimum-weight justifications J in G^{Π} for the set of initial nodes I and the set of goal nodes G correspond to optimal delete-relaxation plans.*

AND/OR graphs are slightly more general than delete-relaxation problems in two respects. First, AND/OR graphs representing planning problems with no deletes do not contain edges connecting two AND nodes or two OR nodes, while AND/OR graphs in general are not constrained in this manner. Second, the weight function in such graphs always has the value 0 for $v \in V_{\text{or}}$, while weight functions for AND/OR graphs in general can take arbitrary real positive values for all nodes.

2.4 Special Cases of the Delete Relaxation

We now state several previously made observations about the delete relaxation, with the objective of highlighting the sources of complexity in the problem. To this end, we attempt to construct from an arbitrary problem Π without deletes a problem that more closely resembles the shortest path problem on a standard directed graph, which is known to be in P(Cormen et al., 2001). In the following, we will often write an operator o without deletes as $\langle Pre(o), Add(o)\rangle$ for simplicity.

Definition 2.9 (Problem equivalence). *A* STRIPS *problem* Π' *is equivalent to a problem* Π *if for each plan* π' *for* Π'*, there exists a plan* π *for* Π *such that* $cost(\pi) = cost(\pi')$*, and* π *can be reconstructed from* π' *in time polynomial in the size of* π'*.*

Proposition 2.10 (Multiple add effects). *For any* STRIPS *problem* $\Pi = \langle F, O, I, G, c\rangle$ *with no deletes, there is an equivalent problem* $\Pi' = \langle F', O', I', G', c'\rangle$ *such that for all* $o' \in O'$, $|Add(o)| = 1$.

Proof. Given $\Pi = \langle F, O, I, G, c\rangle$, let $\Pi' = \langle F', O' \cup O'', I, G, c'\rangle$, where

- $F' = F \cup \{p_o \mid o \in O\}$
- $O' = \{\langle Pre(o), \{p_o\}\rangle \mid o \in O\}$
- $O'' = \bigcup_{o \in O} \{\langle \{p_o\}, \{q\}\rangle \mid q \in Add(o)\}$
- $c'(o) = \begin{cases} 0 & \text{if } o \in O'' \\ c(o') & \text{where } p_{o'} \in Add(o) \text{ otherwise} \end{cases}$

It can then be seen that any plan $\pi' = \langle o_1, \ldots, o_n\rangle$ for Π' can be transformed into a plan for Π by removing all $o_i \in O''$, and replacing all operators $o_j \in O'$ with $o \in O$ such that $p_o \in Add(o_j)$, and that $cost(\pi) = cost(\pi')$. □

Proposition 2.11 (Multiple goal fluents). *For any* STRIPS *problem* $\Pi = \langle F, O, I, G, c\rangle$ *with no deletes, there is an equivalent problem* $\Pi' = \langle F', O', I', G', c'\rangle$ *such that* $|G| = 1$.

Proof. Given $\Pi = \langle F, O, I, G, c\rangle$, let $\Pi' = \langle F',\ O \cup \{\textsc{end}\},\ I,\ \{g\},\ c'\rangle$, where

- $F' = F \cup \{g\}$
- $\textsc{end} = \langle G, g\rangle$
- $c'(o) = \begin{cases} c(o) & \text{if } o \in O \\ 0 & \text{if } o = \textsc{end} \end{cases}$

It can then be seen that any plan π' for Π' must have the structure $\pi' = \langle o_1, \ldots, o_n, \textsc{end}\rangle$, as END is the only action adding the newly introduced goal fluent g, and that $\pi = \langle o_1, \ldots, o_n\rangle$ must constitute a plan for Π with $cost(\pi) = cost(\pi')$. □

Proposition 2.12 (Multiple initial fluents)**.** *For any* STRIPS *problem* $\Pi = \langle F, O, I, G, c\rangle$ *with no deletes, there is an equivalent problem* $\Pi' = \langle F', O', I', G', c'\rangle$ *such that* $|I| = 1$.

Proof. Given $\Pi = \langle F, O, I, G, c\rangle$, let $\Pi' = \langle F',\ O \cup \{\textsc{start}\},\ \{i\},\ G,\ c'\rangle$, where

- $F' = F \cup \{i\}$
- $\textsc{start} = \langle \{i\}, I\rangle$
- $c'(o) = \begin{cases} c(o) & \text{if } o \in O \\ 0 & \text{if } o = \textsc{start} \end{cases}$

It can then be seen that any plan π' for Π' must have the structure $\pi' = \langle \textsc{start}, o_1, \ldots, o_n\rangle$, as START is the only action that can be applied in the new initial state which consists of only the newly introduced fluent i, and that $\pi = \langle o_1, \ldots, o_n\rangle$ must constitute a plan for Π with $cost(\pi) = cost(\pi')$. □

Combining these results, we have that for any delete relaxation problem $\Pi = \langle F, O, I, G, c\rangle$, it is possible to obtain an equivalent problem such that the add effects of all operators are unary, and both the initial state I and the goal state G consist of a single fluent. This problem differs from the Shortest Path Problem (SPP) only in that operators may have multiple preconditions,

$\lvert Pre(o) \rvert$	$\lvert G \rvert$	Problem	Complexity
unlimited	unlimited	Optimal Hyperpath	NP-complete
1	unlimited	Directed Steiner Tree	NP-complete
1	1	Shortest Path	P

Table 2.1: The complexity of optimally solving special cases of the delete relaxation problem.

and this turns out to make the difference in complexity between the SPP which is in P, and the Optimal Hyperpath Problem (OHP) and Directed Steiner Tree Problems (DSTP) which are NP-complete (Table 2.1). Taking into account Proposition 2.11 above, the DSTP can be seen as a special case of the OHP in which a single edge e with multiple source nodes $|s(e)| > 1$ is allowed, and the problem is restricted to looking for paths to the single target node of this edge. While this does not decrease the worst-case complexity of the problem, we include it here as it has been extensively studied in the literature (Proemel and Steger, 2002; Zelikovsky, 1997; Charikar et al., 1998; Robins and Zelikovsky, 2000), and one of the heuristics we present in Chapter 4 is inspired by this problem.

Chapter 3

Exploiting the Independence Assumption

In this chapter we present the independence assumption and planning heuristics based on this assumption as applied to costs, and show how they can be used to obtain relaxed plans. We also show how these heuristics can be seen as minimizing certain measures in a structure obtained from a plan or hyperpath, known in the literature as the *unfolded hyperpath*. We then present the set-additive heuristic and discuss how it differs from these heuristics in its application of the independence assumption, which leads to certain subtleties in its computation.

3.1 The Independence Assumption

In Chapter 2, we discussed why the fundamental source of complexity in optimally solving the delete relaxation problem is the presence of operators with multiple preconditions, which renders it an NP-complete problem. Heuristics therefore commonly employ further relaxations or simplifications in order to solve the problem suboptimally. Since the resulting heuristic values are the costs of non-optimal relaxed plans, they are not admissible, yet they are informative and can be used to quickly find non-optimal solutions to planning problems. One such simplification that has been proposed is the *independence assumption*.

The independence assumption takes various forms in different heuristics, but in general can be seen as the assumption that the optimal cost of, or optimal plan for, a set of fluents Q can be obtained as some function of the optimal costs of, or optimal plans for, each of the individual fluents $q \in Q$. This assumption is incorrect due to interactions between plans that achieve different fluents. In a planning problem with deletes, achieving a set of fluents in conjunction may be more or less expensive than the sum of the costs of achieving each individually. It may be more expensive, for example, to achieve a set of fluents $Q = \{q_1, q_2\}$ if all plans achieving q_1 delete another fluent p, which must be made true again in order to achieve q_2. In contrast, in problems without deletes all interactions are positive, since variables retain the values they are assigned by any action in the plan. The cost of the optimal plan to achieve a set of fluents is therefore bounded by the sum of the costs of the optimal plans for the individual fluents:

$$cost(\pi^{+*}(Q)) \leq \sum_{q \in Q} cost(\pi^{+*}(q)) \tag{3.1}$$

In practice, the optimal plan for a set of fluents may consist of the union of the optimal plans for each of the fluents, the optimal plans for some of the fluents combined with other actions that take advantage of the fluents achieved by these plans, or may not intersect at all with the optimal plans for the individual fluents (Figures 3.1a, 3.1b, 3.1c respectively). Heuristics that employ the independence assumption ignore these interactions, and associate with a set of fluents $Q = \{q_1, \ldots, q_n\}$ a value obtained from a function of the values associated with each q_i. In problems in which the size of all precondition sets and the goal set is 1, the assumption is trivially correct, and heuristics based on the independence assumption are optimal.

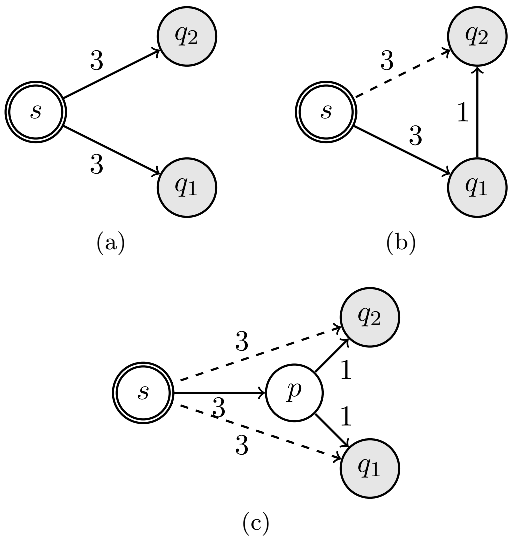


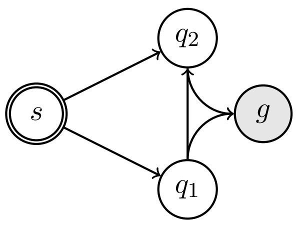
Figure 3.1: $Q = \{q_1, q_2\}$. Optimal plans for q_1, q_2 in isolation are $\{s \to q_1\}$, $\{s \to q_2\}$ respectively in all three problems.

3.2 Unfolded Plans

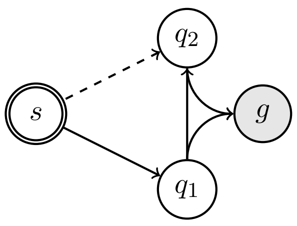
To explain the properties of the solutions found for delete relaxation problem by methods which employ the independence assumption, we briefly discuss *unfolded plans*. Unfolded plans are analogous to unfolded hyperpaths in the directed hypergraph setting (Ausiello et al., 1998).

Definition 3.1 (Unfolded plan)**.** *An unfolded plan U for a problem $\Pi = \langle F, I, O, \{g\}\rangle$, consists of a tree whose nodes are pairs $\langle P, q\rangle$ such that there exists an operator $o \in O$ with $Pre(o) = P$ and $q \in Add(o)$, and has the following properties:*

- *The root node $r = \langle P, g\rangle$ has the goal g as its second value,*
- *For each leaf node $l = \langle P, q\rangle$, $P \subseteq I$,*
- *For each non-leaf node $n = \langle P, q\rangle$, $\cup_{\langle P', q'\rangle \in children(n)}\{q'\} = P$.*



(a) A delete relaxation problem Π^+.



(b) A plan for Π^+.

$$\begin{tikzcd}
 & \langle\{q_1, q_2\}, g\rangle \arrow[dl] \arrow[dr] & \\
\langle\{s\}, q_1\rangle & & \langle\{q_1\}, q_2\rangle \arrow[d] \\
 & & \langle\{s\}, q_1\rangle
\end{tikzcd}$$

(c) The unfolded plan.

Figure 3.2: Folded and unfolded delete relaxation plans.

Figure 3.2 shows a delete relaxation problem along with a relaxed plan for it and the corresponding unfolded plan. Unfolded plans are useful representations of heuristics making use of the independence assumption, as the trees rooted at each node $\langle P, q\rangle$ represent the unfolded plans U_p for each fluent $p \in P$ which are computed independently of one other, and do not interact with the unfolded plans for the other fluents in the set since U does not contain cycles. Given an operator o and unfolded plans U_q for each $q \in \mathit{Pre}(o)$, an unfolded plan U_p can then be obtained for a fluent $p \in \mathit{Add}(o)$ by taking $\langle \mathit{Pre}(o), p\rangle$ to be the root and adding as its children the unfolded plans for each $q \in \mathit{Pre}(o)$. Independence assumption-based heuristics can then be seen as minimizing the cost according to some measure of each of these subtrees independently, and then applying this composition operation to obtain an unfolded plan for the global problem.

Given an unfolded plan for a fluent g, the sequence of operators corresponding to a postordering (Cormen et al., 2001) of the nodes of the tree is a plan for g. However, such a plan may contain multiple instances of a single operator. Since in delete-relaxation problems fluents never become false after being made true, these repetitions are always unnecessary. A closer approximation that constitutes a plan for the problem can therefore be obtained by simply removing all such repetitions from the plan:

Proposition 3.2. *If U is an unfolded plan for $\{g\}$, the set of all operators o such that U contains a node n corresponding to o constitutes a relaxed plan for g.*

3.3 Relaxed Plans From Best Supporters

In practice, unfolded paths which contain the same subtree below all nodes $\langle P, g \rangle$ for which g is the same fluent can be represented in a much more compact manner: it is sufficient to keep a function that maps to each fluent p an operator a_p, called its *best supporter*. The (unfolded) path to any fluent can then be recovered by recursively collecting best supporters of each precondition until fluents that are already present in the initial state are reached. Given a best supporter function, the algorithm shown in Figure 3.3 extracts a plan with at most a single instance of each operator.

As long as the plan represented by the best supporter function is well-founded (*i.e.* does not contain cycles), this algorithm will terminate. Several ways of selecting best supporters have been proposed that minimize different measures on the unfolded path to each fluent. Here we discuss two that have been used with success. In the following definitions $O(p)$ denotes the set $\{o \mid p \in \mathit{Add}(o)\}$.

The Additive Heuristic

The additive heuristic h^{add} applies the independence assumption by recursively estimating the *cost* of a set of fluents as the sum of the costs of each of the individual fluents (Bonet and Geffner, 2001). This is equivalent to making a “worst-case” assumption of independence, in which there is no positive interaction between different fluents, and achieving one fluent makes no progress towards achieving any other fluent. The additive heuristic estimate of the cost of a fluent p is given by:

```
**Input**: A planning problem $\Pi = \langle F, I, O, G \rangle$
**Input**: A best supporter function
        $\texttt{BestSupporter} : F \mapsto O$
**Output**: A relaxed plan $\pi$ or DEAD-END

$\pi \leftarrow \emptyset$
$\mathit{supported} = \emptyset$
$\mathit{to\text{-}support} \leftarrow G$
while $\mathit{to\text{-}support} \neq \emptyset$ do
    Choose $p \in \mathit{to\text{-}support}$
    $\mathit{to\text{-}support} \leftarrow \mathit{to\text{-}support} \setminus \{p\}$
    if $p \notin I$ then
        if $\texttt{BestSupporter}(p) = \mathit{undefined}$ then
            return DEAD-END
        $\pi \leftarrow \pi \cup \{\texttt{BestSupporter}(p)\}$
        $\mathit{supported} \leftarrow \mathit{supported} \cup \{p\}$
        $\mathit{to\text{-}support} \leftarrow$
        $\mathit{to\text{-}support} \cup (\mathit{Pre}(\texttt{BestSupporter}(p)) \setminus \mathit{supported})$
return $\pi$
```

Figure 3.3: The relaxed plan extraction algorithm.

$$h^{\text{add}}(p;s) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s \\ h^{\text{add}}(a_p^{add};s) & \text{otherwise} \end{cases} \tag{3.2}$$

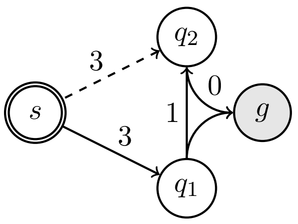
where a_p^{add} denotes the best supporter chosen for fluent p by the h^{add}, and $h^{\text{add}}(a_p^{add};s)$ denotes the cost of applying this action. These are given by the following:

$$a_p^{add} \overset{\text{def}}{=} \text{argmin}_{a \in O(p)} h^{\text{add}}(a;s) \tag{3.3}$$

$$h^{\text{add}}(a;s) \overset{\text{def}}{=} \mathit{cost}(a) + h^{\text{add}}(\mathit{Pre}(a);s) \tag{3.4}$$

$$h^{\text{add}}(Q;s) \overset{\text{def}}{=} \sum_{q \in Q} h^{\text{add}}(q;s) \tag{3.5}$$

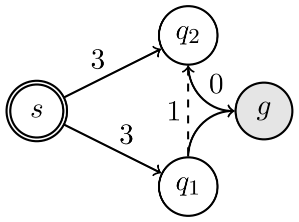
In words, the heuristic estimate of the cost of applying an action is the cost of achieving its set of preconditions, given by the sum of the cost of achieving each, plus the cost of the action itself.



(a) Optimal plan for problem shown in Figure 3.2a, cost 4

$$\begin{tikzcd}
 & \langle\{q_1,q_2\},g\rangle(0) \arrow[dl] \arrow[dr] & \\
\langle\{s\},q_1\rangle(3) & & \langle\{q_1\},q_2\rangle(1) \arrow[d] \\
 & & \langle\{s\},q_1\rangle(3)
\end{tikzcd}$$

(b) Unfolded plan corresponding to optimal plan, additive cost 7



(c) Suboptimal plan for problem shown in Figure 3.2a, cost 6

$$\begin{tikzcd}
 & \langle\{q_1,q_2\},g\rangle(0) \arrow[dl] \arrow[dr] & \\
\langle\{s\},q_2\rangle(3) & & \langle\{s\},q_1\rangle(3)
\end{tikzcd}$$

(d) Unfolded plan corresponding to suboptimal plan, additive cost 6

Figure 3.4: Delete relaxation plans and their corresponding unfolded plans.

Among all of the actions that add a fluent, the action having the lowest heuristic estimate of the cost of application is defined to be its best supporter.

In terms of the unfolded paths discussed above, the $h^{\text{add}}(p; s)$ value is the minimum over all unfolded paths U for p of the sum of the costs of the nodes in U, where the cost of a node is the cost of the operator to which it corresponds. The cost of a single operator may therefore contribute multiple times to the heuristic estimate, behaviour that is known as *overcounting*. While the procedure shown above (Figure 3.3) eliminates this, the relaxed plan that is found is that which minimizes the additive cost of the corresponding unfolded plan, and not the cost of the plan itself, which is not in general equivalent (Figure 3.4).

The Max Heuristic

The additive heuristic is pessimistic about the cost of the delete relaxation and estimates the cost of a set of fluents as the sum of the costs of achieving each fluent individually. This can be seen as the assumption that a relaxed plan achieving one fluent in a set makes no progress towards achieving any of the other fluents, and results in a heuristic estimation that is an upper bound on the optimal cost of the delete relaxation problem. In contrast, the $h^{\max}$ heuristic can be seen as assuming that achieving the most expensive fluent in a set will make all of the other fluents in the set true as "side effects". It then defines the cost of achieving a set of fluents as the cost of achieving the most expensive fluent among them (Bonet and Geffner, 2001):

$$h^{\max}(p; s) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s \\ h^{\max}(a_p^{max}; s) & \text{otherwise} \end{cases} \tag{3.6}$$

Here, a_p^{max} and $h^{\max}(a; s)$ denote the best supporter and estimated cost of applying an action as above, yet are defined differently:

$$a_p^{max} = \text{argmin}_{a \in O(p)} h^{\max}(a; s) \tag{3.7}$$

$$h^{\max}(a; s) \overset{\text{def}}{=} cost(a) + h^{\max}(Pre(a); s) \tag{3.8}$$

$$h^{\max}(Q; s) \overset{\text{def}}{=} \max_{q \in Q} h^{\max}(q; s) \tag{3.9}$$

This change renders the value of the $h^{\max}$ heuristic admissible. In terms of unfolded plans, the value $h^{\max}(p)$ is the minimum over all unfolded plans U for p of the maximum cost path from the root node of U to any one of its leaf nodes, where the cost of such a path is given by the sum of the costs of the operators corresponding to each node in the path.

Using the $h^{\max}$ heuristic for the extraction of relaxed plans was first proposed with a different formulation in the context of the FF planner (Hoffmann and Nebel, 2001). There, it was stated as a combination of the layered *relaxed planning graph* with a plan extraction algorithm enhanced with several secondary heuristics to guide the choice of best supporters. However, upon closer examination it can be seen that the layer at which a fluent first appears in the planning graph is equivalent to the $h^{\max}$ value of the fluent when all operator costs in the problem are set to 1. The *no-ops first* heuristic used by FF to select for each fluent p a supporter which appears at the earliest possible layer therefore ensures that a supporter with minimal $h^{\max}$ value is chosen. The FF heuristic is therefore equivalent to relaxed plan extraction using the $h^{\max}$ heuristic with no costs to choose best supporters.

3.4 The Set-Additive Heuristic

The two heuristics discussed in the previous section make use of the independence assumption for costs to minimize some measure of the cost of the unfolded plan. The overcounting resulting from multiple instances of action nodes in unfolded plans is then eliminated by extracting the corresponding plan. In contrast, the set-additive heuristic h^{sa} applies the independence assumption directly to relaxed plans represented by sets, approximating the optimal relaxed plan for a set of fluents as the union of the relaxed plans for each fluent. The heuristic estimate for a given state is then the cost of the set-additive relaxed plan π^{sa}:

$$h^{\text{sa}}(G;s) \stackrel{\text{def}}{=} cost(\pi^{\text{sa}}(G;s)) \tag{3.10}$$

which is given by the following equations:

$$\pi^{\text{sa}}(p;s) \stackrel{\text{def}}{=} \begin{cases} \{\} & \text{if } p \in s \\ \pi^{\text{sa}}(a_p^{sa};s) & \text{otherwise} \end{cases} \tag{3.11}$$

where

$$a_p^{sa} \stackrel{\text{def}}{=} \text{argmin}_{a \in O(p)} cost(\pi^{\text{sa}}(a;s)) \tag{3.12}$$

$$\pi^{\text{sa}}(a;s) \stackrel{\text{def}}{=} \{a\} \bigcup \pi^{\text{sa}}(Pre(a);s) \tag{3.13}$$

$$\pi^{\text{sa}}(Q;s) \stackrel{\text{def}}{=} \cup_{q \in Q} \pi^{\text{sa}}(q;s) \tag{3.14}$$

and the cost of a relaxed plan is, as always, given by the sum of the costs of the operators it contains:

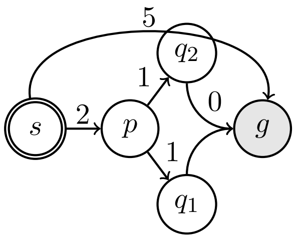
$$cost(\pi) \stackrel{\text{def}}{=} \sum_{a \in \pi} cost(a) \tag{3.15}$$

The intuition behind the set-additive heuristic is that if an operator appears in multiple fluents belonging to a set, only the cost of a single instance of the operator need be counted towards the cost of the overall plan. This intuition is realized by propagating sets representing relaxed plans rather than numeric costs, which allow the operators responsible for the cost of each fluent or subtree in an unfolded plan to be considered when choosing its best supporter. One case in which this information proves to be useful can be seen in Figure 3.5. In this example, h^{sa} chooses as the best supporter for g the action $\{q_1, q_2\} \rightarrow \{g\}$, since in computing the cost of achieving the precondition set $\{q_1, q_2\}$, the cost of the action $\{s\} \rightarrow \{p\}$ is counted only once when the union of the plans for q_1 and q_2 is taken. In contrast, h^{add} considers the cost of the two fluents and comes up with a cost estimate of 6 for the set, which is higher than the correctly calculated cost of the alternative way of achieving g, $\{s\} \rightarrow \{g\}$. However, it is important to note that the values found by h^{sa} are not optimal in general.

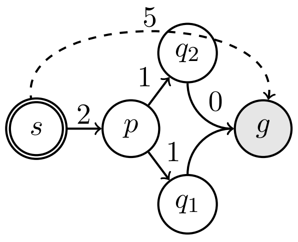
In this context, an alternative way of seeing the regular cost-additive heuristic discussed in Section 3.3 is as the *multiset* additive heuristic. Multisets are set-like objects in which each element is associated with a count denoting how many times it is repeated. When these structures are used rather than regular sets, the union operation above results in a multiset that contains each operator the sum of the number of times that it appears in the plan for each of the fluents contained in a set, and gives the same overcounting behavior seen in the additive heuristic.

Computation

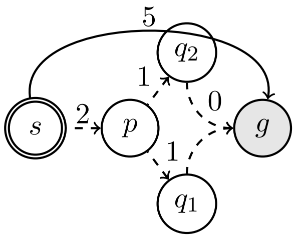
There are two issues that must be considered in the computation of h^{sa}. The first of these is one of efficiency: while the cost of the relaxed plan obtained from the set-additive heuristic is in many cases a more accurate approximation of the optimal cost of the delete relaxation than that of the plan obtained from the best supporters defined by the regular cost-additive heuristic, this comes at the cost of the extra overhead associated with storing sets and performing the costly set union operation to obtain



(a) A delete relaxation problem



(b) Optimal plan found by h^{sa}, cost 4



(c) Subptimal plan found by h^{add}, cost 5

Figure 3.5: Set-additive vs. cost-additive plans.

plans for sets of fluents rather than the much cheaper operation of summing costs. We have found that sorted lists representing relaxed plans work well, as the set union operation can be performed in time linear in the size of the sets. Furthermore, relaxed plans tend to be sparse subsets of the set of operators O of the problem, and structures that grow linearly with its size $|O|$, such as boolean vectors, tend to be inefficient, even if they allow for quick set union implemented as the logical OR operation.

The second and more interesting computational issue concerns the algorithm used to find the fixpoint solution to the recursive equations that define the heuristic. For heuristics such as h^{add} and $h^{\max}$, generalized versions of the Bellman-Ford or Dijkstra algorithms are typically used (Bellman, 1958; Cormen et al., 2001; Dijkstra, 1959; Knuth, 1977; Liu et al., 2002). These algorithms are guaranteed to terminate with the unique fixpoint solution which minimizes the estimated cost of *all* fluents in the problem when the functions used in the equations satisfy a property known as *superiority*, originally defined in the context of the *grammar problem* (Knuth, 1977):

Definition 3.3 (Superior function). *A function $g(x_1, \ldots, x_k) : (\mathbb{R}_0^+)^k \mapsto \mathbb{R}_0^+$ is a superior function if it is monotone nondecreasing in each x_i and if $g(x_1, \ldots, x_k) \geq \max(x_1, \ldots, x_k)$ for all $x_1, \ldots, x_k$.*

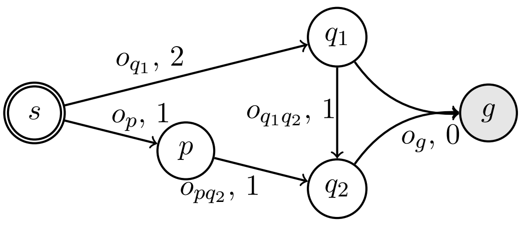
The sum and maximum functions used in the h^{add} and $h^{\max}$ equations are both superior functions. As h^{sa} is defined in terms of sets and not the positive real numbers, we must state an equivalent criterion for functions with non-numeric arguments and a cost function over these arguments:

Definition 3.4 (Generic superior function). *Let A be any set and $c : A \mapsto \mathbb{R}_0^+$ a cost function over that set. A function $g(a_1, \ldots, a_k) : A^k \mapsto A$ is a superior function for cost function c if $c(g(a_1, \ldots, a_k))$ is nondecreasing in each $c(a_i)$ and if $c(g(a_1, \ldots, a_k)) \geq \max(c(a_1), \ldots, c(a_k))$ for all $a_1, \ldots, a_k$.*

In the case of h^{sa}, the function g is $\cup$, the union operator over sets, and the set A is the set of subsets $\mathcal{P}(O)$ of the set of operators of the problem. It is then easy to see that $\cup$ is not a superior function with respect to the sum of the costs of the operators in the set. For an example, let c be the unit cost function $c(A) = |A|$, and consider three sets $B = \{o_1, o_2, o_3\}$, $C = \{o_2, o_3, o_4\}$, $D = \{o_5, o_6\}$. We then have that $c(D) < c(C)$, yet $c(g(B, C)) < c(g(B, D))$. The fact that the set union operation does not satisfy the criteria for being a generic superior function is not surprising: if it did, the equations defining h^{sa} could be minimized simultaneously, in which case it would be possible to compute the cost of a minimal cost plan for a set of fluents Q by combining the minimal plans for the members of the set $q \in Q$. This is equivalent to the independence assumption that we already know to be false.

It is easy to construct a problem in which the Bellman-Ford and Dijkstra algorithms find solutions to the h^{sa} equations that are either non-minimal for some variable or inconsistent.[1] Con-

[1]We thank Malte Helmert for pointing this issue out to us (private communication, 2009).



(a) A delete relaxation problem

Figure 3.6: A delete relaxation problem illustrating the quirks of h^{sa}.

	s	p	q_1	q_2	g
0	$\{\}$				
1	$\{\}$		$\{o_{q_1}\}$ (2)		
2	$\{\}$		$\{o_{q_1}\}$ (2)	$\{o_{q_1}, o_{q_1q_2}\}$ (3)	
3	$\{\}$		$\{o_{q_1}\}$ (2)	$\{o_{q_1}, o_{q_1q_2}\}$ (3)	$\{o_{q_1}, o_{q_1q_2}, o_g\}$ (3)
4	$\{\}$	$\{o_p\}$ (1)	$\{o_{q_1}\}$ (2)	$\{o_{q_1}, o_{q_1q_2}\}$ (3)	$\{o_{q_1}, o_{q_1q_2}, o_g\}$ (3)
5	$\{\}$	$\{o_p\}$ (1)	$\{o_{q_1}\}$ (2)	$\{o_p, o_{pq_2}\}$ (2)	$\{o_{q_1}, o_{q_1q_2}, o_g\}$ (3)

Table 3.1: Possible sequence of updates with the Bellman-Ford algorithm. Solution is inconsistent as $\pi^{\text{sa}}(g) \neq \{o_g\} \cup \pi^{\text{sa}}(q_1) \cup \pi^{\text{sa}}(q_2)$.

sider the problem shown in Figure 3.6a. If the Bellman-Ford algorithm is used, certain sequences of updates result in an inconsistent solution (see Table 3.1), while if Dijkstra's algorithm is used, plans found for fluents are not necessarily minimal (see Table 3.2). However, Dijkstra's algorithm is guaranteed to find the minimal *consistent* solution, and is the algorithm we use to compute the value of h^{sa}.

3.5 Experimental Results

We evaluate the performance of the heuristics discussed above in several respects, asking which produce better approximations of the optimal cost of the delete relaxation, which are most efficient in terms of computational effort, and finally which result in the best overall planning performance in terms of coverage and plan

	s	p	q_1	q_2	g
0	$\{\}$				
1	$\{\}$	$\{o_p\}$ (1)			
2	$\{\}$	$\{o_p\}$ (1)	$\{o_{q_1}\}$ (2)	$\{o_p, o_{pq_2}\}$ (2)	
3	$\{\}$	$\{o_p\}$ (1)	$\{o_{q_1}\}$ (2)	$\{o_p, o_{pq_2}\}$ (2)	$\{o_{q_1}, o_p, o_{pq_2}\}$ (4)

Table 3.2: Sequence of updates with Dijkstra's algorithm. $\pi^{\text{sa}}(g)$ has cost 4, compared to 3 in the inconsistent solution found with the Bellman-Ford algorithm.

quality. In addition to the relaxed plans extracted using the best supporters for each fluent as determined by the $h^{\max}$ and h^{add} heuristics, and the relaxed plan given by the h^{sa} heuristic, we consider the relaxed plan resulting from a basic implementation of the h^{ff} heuristic, in which best supporters with minimum level in the planning graph, or equivalently, minimum $h^{\max}$ value when costs are treated as uniform, are chosen. The secondary heuristics described in Hoffmann and Nebel (2001), such as the selection of supporters with a minimum number of delete effects, are not used. As our test domains, we use the set of satisficing planning instances from the most recent International Planning Competition (IPC6). All experiments discussed below were run on Xeon Woodcrest computers with clock speeds of 2.33 GHz, using a 2GB memory limit and a time cutoff of 1800 seconds when appropriate.

Quality of Π^+ Approximation

We first consider which of the heuristics discussed above is best able to approximate the optimal cost of the delete relaxation. Table 3.3 shows the number of problems in which each heuristic generates the lowest and second-lowest cost relaxed plans for the initial states of the thirty instances of each domain. Table 3.4 evaluates the heuristics on a larger number of states, and shows the proportion of states for which each heuristic produces the lowest cost plan. In both tables, individual entries sum to more than the total number of instances, or more than 100%, when two or more heuristics generate relaxed plans with the same cost.

Domain	h^{ff}	$\pi(h^{\max})$	$\pi(h^{\text{add}})$	h^{sa}
cybersec (30)	9 / 14	15 / 13	16 / 11	30 / 0
elevators (30)	2 / 8	23 / 7	17 / 11	15 / 10
openstacks (30)	0 / 30	30 / 0	30 / 0	30 / 0
parcprinter (30)	7 / 16	23 / 4	25 / 5	29 / 0
pegsol (30)	4 / 16	19 / 11	4 / 17	25 / 5
scanalyzer (30)	14 / 3	17 / 9	27 / 2	28 / 1
sokoban (30)	5 / 19	21 / 8	21 / 9	22 / 7
transport (30)	1 / 11	19 / 5	12 / 11	4 / 7
woodworking (30)	1 / 10	16 / 9	8 / 18	17 / 2
total (270)	43 / 127	183 / 66	160 / 84	200 / 32

Table 3.3: Quality of independence assumption based heuristics. Entries indicate number of times each heuristic found lowest/second lowest cost relaxed plan out of the four heuristics. The heuristics were computed for the initial state of each of the 30 problems per domain.

Domain	h^{ff}	$\pi(h^{\max})$	$\pi(h^{\text{add}})$	h^{sa}
cybersec (300000)	0.40	0.78	0.75	0.84
elevators (300000)	0.05	0.83	0.52	0.46
openstacks (221684)	0.02	1.00	1.00	1.00
parcprinter (229963)	0.04	0.67	0.83	0.84
pegsol (214143)	0.08	0.68	0.11	0.71
scanalyzer (230183)	0.32	0.22	0.56	0.52
sokoban (248638)	0.14	0.60	0.59	0.78
transport (210700)	0.15	0.60	0.23	0.13
woodworking (183331)	0.04	0.50	0.24	0.63
total (2138642)	0.15	0.67	0.55	0.66

Table 3.4: Quality of independence assumption based heuristics. Entries indicate proportion of states for which each heuristic found lowest cost relaxed plan out of the four heuristics. Number of states for which the heuristics were evaluated is indicated next to each domain in parenthesis.

Domain	h^{ff}	$\pi(h^{\max})$	$\pi(h^{\mathrm{add}})$	h^{sa}
cybersec (300000)	1.00	0.66	0.66	0.65
elevators (290000)	1.00	0.62	0.67	0.68
openstacks (212031)	1.00	0.37	0.37	0.37
parcprinter (229963)	1.00	0.88	0.88	0.88
pegsol (214128)	1.00	0.75	0.98	0.74
scanalyzer (230183)	1.00	0.96	0.94	0.94
sokoban (248638)	1.00	0.92	0.93	0.91
transport (210700)	1.00	0.87	1.03	1.11
woodworking (183331)	1.00	0.92	0.93	0.92
total (2118974)	1.00	0.76	0.81	0.79

Table 3.5: The average proportion of the costs of the relaxed plans found by different heuristics to those found by h^{ff}. Number of states for which the heuristics were evaluated is indicated next to each domain in parenthesis.

Finally, in Table 3.5, we show the average ratio of the cost of the relaxed plan generated by each heuristic to that generated by the h^{ff} heuristic which ignores costs. For this experiment and the previous one, the states to be evaluated in each problem are generated by blind search from the initial state.

Our first observation is a rather obvious one: taking into account the costs of actions when choosing supporters generally results in lower cost relaxed plans. The relaxed plans found by h^{ff} are consistently worse than those found by the other heuristics. h^{ff} finds the lowest cost plan only for only 43 out of the 270 initial states considered, compared to 160 for the second-worst heuristic, and for only 15% of a larger number of evaluated states, compared to 55% for the second-worst heuristic. It finds worse relaxed plans than the other heuristics in almost all cases when considered on a per-domain basis as well, outperforming only the $h^{\max}$ and h^{sa} heuristics on a single domain each, scanalyzer and transport respectively. In terms of the relative costs of the plans found, the picture is similar, with the average cost of the relaxed plan found by other heuristics being higher in only one domain, transport, for the h^{sa} and h^{add} heuristics.

Comparing $h^{\max}$, h^{add}, and h^{sa}, it is hard to choose a clear winner. Each heuristic generates lower cost plans in the largest number of states for some domain, in the case of h^{max} the elevators and transport domains, in the case of h^{add} the scanalyzer domain, and in the case of h^{sa} the cybersec, parcprinter, pegsol, sokoban, and woodworking domains. The plans generated by all three heuristics for the openstacks domain have the same cost. In general, h^{sa} outperforms h^{add}, generating the lowest-cost plans in up to 60% more states than h^{add} in five domains, and generating the lowest-cost plans in a slightly lower number of states in three. In the three domains in which h^{add} generates the lowest-cost plans in more states, the difference is never larger than 10% of the states evaluated. Between the h^{max} and h^{sa} heuristics, the picture is more varied. In the elevators and transport domains, h^{max} generates the best relaxed plans in 37% and 47% more of the instances respectively. Yet in the other six domains for which there are variations in relaxed plan cost, h^{sa} generates the best plans for between 3% and 30% percent more states. This leads to the two heuristics performances being approximately equal when considered over the full set of domains, yet as seen above there are large variations on a per-domain basis.

When considered through the lens of relative plan costs, the picture is similar, with h^{sa} generating plans of slightly less cost than those generated by h^{add} on average, and h^{max} generating better plans than both, principally due to its performance in the transport and elevators domains. Note, however that this type of comparison is less effective at distinguishing between the heuristics, and values tend to be more uniform. For an example, consider the parcprinter domain, in which h^{sa} generates the best relaxed plans in 84% of the considered states compared to 83% for h^{add}, 67% for h^{max}, and 4% for h^{ff}. When considered in terms of relative plan costs, all three of the cost based heuristics generate plans that are on average 88% as costly as those of h^{ff}. Several factors may contribute to this: the differences in cost between the best relaxed plan found and the second or third best plans may be small, and the heuristics may tend to find better plans for different states, resulting in similar average values.

Domain	h^{ff}	$\pi(h^{\max})$	$\pi(h^{\text{add}})$	h^{sa}
cybersec	718	874	1887	109
elevators	1142	1536	1202	303
openstacks	113	898	84	27
parcprinter	6610	5966	5999	3808
pegsol	12888	10748	12293	5830
scanalyzer	52	113	70	39
sokoban	2568	2547	2387	475
transport	536	569	320	48
woodworking	237	260	369	152
total	1419	2118	1726	496

Table 3.6: Average number of heuristic evaluations performed per second for independence assumption based heuristics. To minimize variance, only problems solved with > 1000 heuristic evaluations were considered.

Computational Cost of Heuristics

We now look at the issue of the computational effort required to compute the value of each heuristic. We expect the computation of h^{ff}, $h^{\max}$, and h^{add} to be faster than that of h^{sa} due to two factors. The first of these is that the first three heuristics propagate costs rather than sets of actions, and calculating the cost of an action by taking the maximum or sum of the costs of a set of precondition fluents is less computationally intensive than the union operation over sets. Furthermore, as the domain size increases, the number of such operations that must be performed in each state grows for all heuristics, yet in the case of h^{sa}, the difficulty of the operation grows as well, as the sets representing relaxed plans become larger. The second factor is that in order to compute a consistent solution to the set-additive heuristic, the associated equations must be solved with a variant of Dijkstra's algorithm, in which updates are ordered in a priority queue. The overhead of inserting the updates to be applied in the priority queue then implies further overhead. These issues are discussed in more detail in Section 3.4.

Table 3.6 shows the number of heuristic evaluations performed per second by each of the heuristics. In order to reduce variance in the calculation resulting from inaccurate measurement of shorter time periods, only problems that were solved with more than 1000 evaluations were considered. Taking into consideration the factors mentioned above, it is not surprising that the computation of the set-additive heuristic is on average a factor of 2–10 times slower than the computation of the other three heuristics. Evaluation times for the other heuristics exhibit some variation, but are largely similar across the set of domains considered.

Coverage

In this section we consider the overall planning performance of the heuristics in terms of coverage, or the number of problems solved by each heuristic. Results in this section were obtained using a greedy best first search (GBFS) algorithm with delayed evaluation and two open lists, one of which contains only nodes resulting from *helpful actions*, defined as the set of actions in the relaxed plan applicable in the current state. The algorithm is our own implementation of the algorithm used in the winning planner of the satisficing track of the most recent IPC, LAMA (Richter and Westphal, 2010). In choosing a node to be expanded, we alternate between picking a node from the open list containing nodes resulting from helpful actions, and the regular open list, giving no priority to the former. We consider the four heuristics discussed above in three different configurations, (a) computing the heuristics using the operator costs stated in the problem (except in the case of h^{ff}) and taking the cost of the resulting relaxed plan, (b) as (a), but using an added cost for zero cost actions (discussed below), and (c), computing the relaxed plans using costs, but using the size of the relaxed plan $|\pi|$, rather than the cost $cost(\pi)$ as the heuristic value.

It has previously been observed that optimizing relaxed plans for cost rather than size can often be detrimental to coverage (Richter and Westphal, 2010). Here we confirm this observation. Using the *cost* of the relaxed plan generated by the h^{ff} as the heuristic value of the state, more problems are solved than with any other heuristic in six out of the nine domains considered (Table 3.7). The three other domains in which some other

Domain	h^{ff}	$\pi(h^{\text{max}})$	$\pi(h^{\text{add}})$	h^{sa}
cybersec (30)	8	3	5	7
elevators (30)	12	8	8	8
openstacks (30)	15	12	12	10
parcprinter (30)	17	16	16	18
pegsol (30)	29	30	30	30
scanalyzer (30)	26	25	22	21
sokoban (30)	28	27	27	25
transport (30)	12	11	14	14
woodworking (30)	29	28	28	28
total (270)	176	160	162	161

Table 3.7: Coverage for independence assumption based heuristics. The number of problems in each domain is indicated in parenthesis.

Domain	h^{ff}	$\pi(h^{\text{max}})$	$\pi(h^{\text{add}})$	h^{sa}
cybersec (30)	18	10	14	16
elevators (30)	12	11	10	8
openstacks (30)	27	27	30	25
parcprinter (30)	17	16	16	18
pegsol (30)	29	29	30	30
scanalyzer (30)	27	26	21	22
sokoban (30)	26	27	27	23
transport (30)	12	11	14	14
woodworking (30)	29	28	28	27
total (270)	197	185	190	183

Table 3.8: Coverage for independence assumption based heuristics when the cost of all zero cost actions is set to be 0.01. The number of problems in each domain is indicated in parenthesis.

heuristic is able to solve more problems are the parcprinter domain, in which h^{sa} solves one more problem than h^{ff}, pegsol, in which all three other heuristics solve one more problem, and transport, in which h^{sa} and h^{add} both are able to solve two more problems than h^{ff}. When the total number of problems solved is considered, h^{ff} does significantly better, solving 14 more problems than its closest competitor, h^{add}.

Domain	$\|h^{\mathrm{ff}}\|$	$\|\pi(h^{\max})\|$	$\|\pi(h^{\mathrm{add}})\|$	$\|h^{\mathrm{sa}}\|$
cybersec (30)	14	15	20	19
elevators (30)	26	23	24	28
openstacks (30)	30	25	30	28
parcprinter (30)	22	23	22	23
pegsol (30)	29	30	30	30
scanalyzer (30)	26	22	21	22
sokoban (30)	28	26	26	26
transport (30)	13	13	17	16
woodworking(30)	21	27	24	26
total (270)	209	204	214	218

Table 3.9: Coverage for independence assumption based heuristics when plan size rather than plan cost is used. The number of problems in each domain is indicated in parenthesis.

One issue that we have noted in optimizing for plan cost rather than plan length is that operators with zero cost can lead to heuristic plateaus, in which such an operator decreases the number of operators in the relaxed plan for the resulting state, thus advancing towards the goal, but not its cost. In order to test the impact of this, we modify the heuristic computation slightly by assigning to each operator with zero cost a *base cost*, thus ensuring that the removal of an operator from a plan also results in a decrease in the cost of the plan. The results of using a base cost of 0.01 are shown in Table 3.8. We observe that the use of this base cost improves coverage across the board, increasing the total number of problems solved by approximately 20 for each heuristic. Furthermore, this addition also decreases the gap between the results for h^{ff} and the cost based heuristics, with h^{ff} solving the most problems in only four domains, compared to six in the previous experiment, and the gap in the number of problems solved compared to the next best heuristic shrinking to 7, from 14.

Finally, when the cost of the relaxed plan is completely discarded and its *size* $|\pi|$ is used as a heuristic estimate, coverage results are further improved (Table 3.9). Note that costs are not discarded during the computation of the heuristics, so the heuristic computation still attempts to find low cost, rather than size, relaxed plans. Interestingly, the increase in coverage across the different heuristics is not uniform in this case and search using the additive and set-additive heuristics benefits the most. In fact, using the size of the relaxed plans generated by h^{add} and h^{sa} as a heuristic results in both configurations solving more problems than with the h^{ff} heuristic, in the case of h^{add} 5 more, and in the case of h^{sa} 9 more.

Node Expansions

We now consider the number of nodes expanded by each heuristic in order to find a solution. The three configurations used are the same as those discussed above. In our evaluation, we consider only the problems which were solved using all four of the heuristics, and compare the number of nodes expanded to the number of nodes expanded using the h^{ff} heuristic. Each entry in the following tables is then of the form +x/-y, indicating that of the commonly solved problems, the heuristic solved x with more expansions than h^{ff}, and y with less. In the first configuration, using the costs of relaxed plans as the heuristic value, the three cost based heuristics turn out to be much less informative than h^{ff} in terms of goal directedness (Table 3.10). In almost all of the domains, they evaluate a greater number of nodes to find a solution in more problems than the other way around, with large differences especially in the pegsol, woodworking, and openstacks domains. The woodworking domain is a particularly egregious in this respect.

When a base cost of 0.01 is added to the cost of all zero cost actions, results are somewhat improved, with both the h^{max} and h^{sa} heuristics solving problems with approximately the same number of node expansions as the h^{ff} heuristic (Table 3.11).

In the third configuration, costs of actions are ignored by the search algorithm, which instead takes the size of the relaxed plan to be the heuristic value. This is especially detrimental to the cost based heuristics, and results are similar to the first configuration. In almost all domains, the number of problems solved with a greater number of expansions is higher than the number of problems solved with fewer (Table 3.12). The situa-

Domain	h^{ff}	$\pi(h^{\max})$	$\pi(h^{\text{add}})$	h^{sa}
cybersec (2)	+0/-0	+0/-1	+1/-1	+1/-1
elevators (5)	+0/-0	+2/-3	+2/-3	+2/-3
openstacks (10)	+0/-0	+9/-1	+9/-1	+8/-2
parcprinter (15)	+0/-0	+9/-4	+9/-4	+7/-7
pegsol (29)	+0/-0	+17/-10	+15/-12	+18/-10
scanalyzer (20)	+0/-0	+11/-9	+16/-4	+15/-5
sokoban (25)	+0/-0	+10/-15	+10/-15	+10/-15
transport (11)	+0/-0	+5/-6	+1/-10	+1/-10
woodworking (25)	+0/-0	+18/-3	+18/-3	+18/-5
total (142)	+0/-0	+81/-52	+81/-53	+80/-58

Table 3.10: Number of node expansions when using the cost of the generated relaxed plan as the heuristic function. An entry of the form +x/-y indicates that the heuristic solved x problems with more node expansions than h^{ff}, and y problems with fewer. Results are shown only for problems solved with all four heuristics. The number of such problems is indicated in parenthesis.

Domain	h^{ff}	$\pi(h^{\max})$	$\pi(h^{\text{add}})$	h^{sa}
cybersec (8)	+0/-0	+1/-4	+2/-6	+5/-3
elevators (7)	+0/-0	+3/-4	+3/-4	+4/-3
openstacks (20)	+0/-0	+2/-18	+14/-6	+10/-10
parcprinter (15)	+0/-0	+9/-4	+9/-4	+7/-7
pegsol (29)	+0/-0	+18/-10	+14/-14	+12/-16
scanalyzer (20)	+0/-0	+11/-9	+16/-4	+15/-5
sokoban (23)	+0/-0	+9/-14	+7/-16	+6/-17
transport (11)	+0/-0	+5/-6	+1/-10	+1/-10
woodworking (24)	+0/-0	+17/-3	+17/-3	+17/-5
total (157)	+0/-0	+75/-72	+83/-67	+77/-76

Table 3.11: Number of node expansions when using the cost of the generated relaxed plan as the heuristic function, with an added base cost of 0.01 for all zero cost actions. An entry of the form +x/-y indicates that the heuristic solved x problems with more node expansions than h^{ff}, and y problems with fewer. Results are shown only for problems solved with all four heuristics. The number of such problems is indicated in parenthesis.

Domain	$\lvert h^{\text{ff}}\rvert$	$\lvert\pi(h^{\max})\rvert$	$\lvert\pi(h^{\text{add}})\rvert$	$\lvert h^{\text{sa}}\rvert$
cybersec (10)	+0/-0	+4/-3	+3/-7	+5/-5
elevators (23)	+0/-0	+17/-5	+10/-13	+9/-14
openstacks (23)	+0/-0	+22/-0	+23/-0	+21/-2
parcprinter (22)	+0/-0	+17/-3	+17/-3	+12/-9
pegsol (29)	+0/-0	+10/-17	+12/-13	+9/-18
scanalyzer (19)	+0/-0	+9/-10	+11/-8	+8/-11
sokoban (26)	+0/-0	+19/-7	+17/-9	+15/-11
transport (12)	+0/-0	+11/-1	+0/-12	+0/-12
woodworking (19)	+0/-0	+11/-3	+11/-3	+13/-4
total (183)	+0/-0	+120/-49	+104/-68	+92/-86

Table 3.12: Number of node expansions when using the size of the generated relaxed plan as the heuristic function. An entry of the form +x/-y indicates that the heuristic solved x problems with more node expansions than h^{ff}, and y problems with fewer. Results are shown only for problems solved with all four heuristics. The number of such problems is indicated in parenthesis.

tion is particularly bad in the openstacks domain, in which the cost based heuristics require more evaluations in almost all of the instances.

Plan Cost

Here we evaluate the heuristics from the perspective of the cost of the plans that each produce. As in the previous section, the evaluation method is to compare the number of higher/lower cost plans produced compared to the h^{ff} heuristic which ignores costs, considering only the problems which are solved with all four heuristics. In the first configuration, in which the cost of the relaxed plan is used with no added values for zero cost actions, results are greatly improved across the board with the use of costs, with all three heuristics that take into account cost in their computation producing more lower cost plans than higher cost plans compared to h^{ff}(Table 3.13). The best results are obtained by the set-additive heuristic, which produces 78 lower cost plans and 35 higher cost plans. The costs of the generated

Domain	h^{ff}	$\pi(h^{\text{max}})$	$\pi(h^{\text{add}})$	h^{sa}
cybersec (2)	+0/-0	+0/-0	+0/-0	+1/-0
elevators (5)	+0/-0	+2/-2	+3/-2	+3/-2
openstacks (10)	+0/-0	+0/-9	+0/-9	+0/-9
parcprinter (15)	+0/-0	+0/-3	+0/-3	+0/-3
pegsol (29)	+0/-0	+4/-15	+3/-7	+4/-17
scanalyzer (20)	+0/-0	+10/-9	+11/-8	+6/-14
sokoban (25)	+0/-0	+5/-13	+4/-14	+6/-14
transport (11)	+0/-0	+6/-5	+0/-11	+0/-11
woodworking (25)	+0/-0	+12/-9	+12/-9	+15/-8
total (142)	+0/-0	+39/-65	+33/-63	+35/-78

Table 3.13: Higher/lower cost plans when using the cost of the generated relaxed plan as the heuristic function. An entry of the form +x/-y indicates that the heuristic found higher cost plans than h^{ff} for x problems, and lower cost plans for y problems. Results are shown only for problems solved with all four heuristics. The number of such problems is indicated in parenthesis.

plans are lower for almost all of the domains, with the single exception being the woodworking domain, which was also an outlier in terms of the previous criteria considered. In particular, the woodworking domain is responsible for slightly less than half of the worse cost plans produced by the h^{sa}, and approximately a third of the worse cost plans produced by h^{add} and h^{max}. The use of cost based heuristics improves the resulting plans to a large extent particularly in the pegsol, scanalyzer, and sokoban domains.

In the second configuration, in which an added base cost is used for zero cost actions, the differences in plan quality are less (Table 3.14). h^{add} and h^{max} continue to generate more plans of higher cost than lower cost when compared to h^{ff}, with the domains showing particular improvement being the same as those in the previous configuration. One interesting deviation from the previous results is the performance of h^{add} in the openstacks domain, in which it generates 20 plans of lower cost compared to h^{ff}, and no plans of higher cost. The woodworking domain continues to play a large role in the overall results, being responsible for approximately a third of the higher cost plans in the

Domain	h^{ff}	$\pi(h^{\text{max}})$	$\pi(h^{\text{add}})$	h^{sa}
cybersec (8)	+0/-0	+0/-0	+0/-0	+3/-0
elevators (7)	+0/-0	+7/-0	+4/-2	+7/-0
openstacks (20)	+0/-0	+11/-5	+0/-20	+8/-9
parcprinter (15)	+0/-0	+0/-3	+0/-3	+0/-3
pegsol (29)	+0/-0	+2/-14	+5/-7	+5/-14
scanalyzer (20)	+0/-0	+10/-9	+11/-8	+6/-14
sokoban (23)	+0/-0	+6/-11	+5/-11	+7/-12
transport (11)	+0/-0	+6/-5	+0/-11	+0/-11
woodworking (24)	+0/-0	+11/-9	+11/-9	+14/-8
total (157)	+0/-0	+53/-56	+36/-71	+50/-71

Table 3.14: Higher/lower cost plans when using the cost of the generated relaxed plan as the heuristic function, with an added base cost of 0.01 for all zero cost actions. An entry of the form +x/-y indicates that the heuristic found higher cost plans than h^{ff} for x problems, and lower cost plans for y problems. Results are shown only for problems solved with all four heuristics. The number of such problems is indicated in parenthesis.

results for the h^{sa} and h^{add} heuristics.

Finally, when the size of the relaxed plan is used to generate heuristic values rather than its cost, the advantage enjoyed by the cost sensitive heuristics is nullified for both h^{sa} and h^{max} (Table 3.15). Interestingly, the performance of the h^{add} heuristic in this configuration is greatly improved compared to previous configurations, and it generates 80 plans of better cost than those generated by h^{ff}, compared to only 44 plans of worse cost. Similarly to when the heuristics are considered in terms of the number of node expansions required to find a solution, the openstacks domain is a particularly bad case, with the combination of cost sensitive heuristics and relaxed plan size as a heuristic function leading to h^{sa} and h^{max}generating worse plans in almost all of the instances considered.

Domain	$\lvert h^{\text{ff}}\rvert$	$\lvert\pi(h^{\max})\rvert$	$\lvert\pi(h^{\text{add}})\rvert$	$\lvert h^{\text{sa}}\rvert$
cybersec (10)	+0/-0	+0/-3	+0/-3	+1/-3
elevators (23)	+0/-0	+11/-12	+6/-17	+15/-7
openstacks (23)	+0/-0	+21/-1	+2/-10	+23/-0
parcprinter (22)	+0/-0	+10/-10	+11/-9	+9/-10
pegsol (29)	+0/-0	+8/-8	+3/-4	+11/-7
scanalyzer (19)	+0/-0	+7/-7	+4/-9	+4/-12
sokoban (26)	+0/-0	+8/-11	+8/-12	+14/-8
transport (12)	+0/-0	+5/-6	+2/-10	+1/-11
woodworking (19)	+0/-0	+9/-5	+8/-6	+7/-9
total (183)	+0/-0	+79/-63	+44/-80	+85/-67

Table 3.15: Higher/lower cost plans when using the size of the generated relaxed plan as the heuristic function. An entry of the form +x/-y indicates that the heuristic found higher cost plans than h^{ff} for x problems, and lower cost plans for y problems. Results are shown only for problems solved with all four heuristics. The number of such problems is indicated in parenthesis.

3.6 Related Work

Since the independence assumption was first stated explicitly and used to obtain a heuristic value from the delete relaxation, much work has gone into developing new heuristics that explicitly or implicitly depend on it and exploring their properties. Here we briefly review some of this work.

The additive heuristic h^{add} used in the HSP planner is the first heuristic to be stated in terms of a solution to the delete relaxation of a problem, and the first to explicitly make use of the independence assumption (Bonet and Geffner, 2001). The heuristic value used is the sum of the costs of the goals in the problem as discussed in Section 3.3, rather than an explicit relaxed plan described by the set of operators belonging to it. As a result, the heuristic suffers from the overcounting problem, due to which the cost of an operator could count more than once towards the heuristic estimate.

The relaxed planning graph heuristic h^{ff} adapts the original planning graph (Blum and Furst, 1995) to the delete relaxation by dropping the delete information encoded in the form of mutexes in the graph (Hoffmann and Nebel, 2001). The explicit graph representation in which each fluent in the problem is marked with an operator adding it allows the extraction of a relaxed plan in the form of a set in which each operator appears at most once, thereby eliminating the problem of overcounting. However, propagation in the relaxed planning graph as originally defined can not take into account the costs of operators, considering only the level, or number of parallel plan steps, required to achieve a fluent, and is therefore not suitable to planning problems in which it is important to find low-cost plans. The explicit representation of the relaxed plan allows the extraction of helpful actions, defined in the FF planner as those operators applicable in the current state adding some precondition of an operator in the relaxed plan.

The SAPA planner uses several variations of additive and max propagation in the relaxed planning graph followed by relaxed plan extraction as in the FF planner (Do and Kambhampati, 2003). The LAMA planner proposed the use of relaxed plan extraction with best supporters as defined by the additive heuristic independently of our work, and its authors have conducted a thorough analysis of various schemes for setting the costs of actions (Richter and Westphal, 2010). As we do here, they conclude that constructing relaxed plans in a cost-sensitive manner and using their costs rather than sizes as heuristic values, while increasing plan quality, is greatly detrimental to coverage. They also investigate a strategy similar to ours to address the issue of zero-cost operators resulting in heuristic plateaus, summing the cost and size of the relaxed plan to obtain heuristic values. In their setting, this strategy does not result in the coverage increase seen here. We speculate that this difference is due to the fact that in their pure cost version, they break heuristic ties by preferring the state resulting in the plan with smaller size, which appears to be sufficient to give the increase in coverage.

A number of works have focused on a more general framework for describing the theoretical properties of heuristics based on the independence assumption. In the setting of optimal directed hyperpath finding, this has taken the form of investigating the types of cost measures for which optimal hyperpaths can be found in polynomial time (Ausiello et al., 1998). These cost measures turn out to encompass the h^{add} and h^{max} heuristics as special cases. Fuentetaja et al. (2008) also give a general algorithm for computing independence-assumption based heuristics in the relaxed planning graph, with different aggregation functions acting as its building blocks. They show that the heuristics presented here can be computed by various instantiations of their algorithm.

3.7 Conclusions

We have investigated the use of the independence assumption as applied to costs to obtain suboptimal relaxed plans. We have shown that any acyclic best supporter function which assigns to each fluent in the problem an operator that supports it can be used to generate these plans, and used this fact to build upon the h^{add} and h^{ff} heuristics to obtain new heuristics that integrate the advantages of both: namely, sensitivity to costs, the elimination of the overcounting behaviour seen in h^{add}, and the ability to obtain an explicit relaxed plan from which search control information in the form of helpful actions can be extracted. The best supporter functions we use are defined declaratively in terms of recursive equations rather than algorithmically in terms of the planning graph, leading to a better understanding of their properties and a reduction in the space requirements of the algorithms required to compute them. In addition, we have related the independence assumption to the unfolded plan structure, previously investigated in the context of optimal directed hyperpath finding, giving new insight into the properties of heuristics that rely on it.

We have also introduced the set-additive heuristic that moves away from the independence assumption in the context of costs and instead applies it to relaxed plans for fluents, leading to relaxed plans whose costs are a tighter upper bound on the optimal cost of the delete relaxation.

Chapter **4**

Beyond the Independence Assumption

In this chapter we explore methods for solving the delete relaxation or improving existing solutions that instead of assuming independence, attempt to find plans that are globally better. We present first an improvement scheme based on certain properties of Steiner trees, and give an experimental evaluation of the resulting heuristic.

4.1 Steiner Trees and the Delete Relaxation

The Steiner Tree (ST) problem is a graph problem similar to the better known Minimum Spanning Tree (MST) problem. While in the MST problem the goal is to find a minimum weight tree that spans all the nodes in a graph (Cormen et al., 2001), in the ST problem, only a subset of the full set of nodes of the graph must be spanned:

Definition 4.1 (Steiner tree problem). *Given a graph $G = \langle V, E\rangle$ and a set of* target nodes $T \subseteq V$ *which is a subset of the full set of nodes, the Steiner tree problem is the problem of* *finding a tree S with minimum weight that spans all the nodes in T.*

When the set of target nodes T is equal to the full set of nodes in the graph, the ST problem is equivalent to the MST problem. The MST problem can be solved in polynomial time by greedy approaches such as Prim's or Kruskal's algorithms (Cormen et al., 2001), yet the ST problem is in general intractable (Proemel and Steger, 2002) and was in fact one of the original 21 NP-complete problems presented by Karp (1972). The associated optimization problem of finding the lowest cost tree spanning T is therefore NP-hard. Here, we refer to such a minimum cost tree as a Steiner tree (ST), and to a tree which spans the set of nodes T while not necessarily having minimal weight as a candidate Steiner tree (CST).

The ST problem can be encoded straightforwardly as a planning problem without deletes:

Definition 4.2 (Delete relaxation encoding of ST problem). *Given an ST problem S specified by a graph $G = \langle V, E\rangle$ and a set of target nodes $T \subseteq V$, let $P_S = \langle F, I, O, G\rangle$, where*

- $F = V$
- $I = n_0$, *where* $n_0 \in T$ *is any node in* T
- $O = \bigcup_{\langle n,n'\rangle \in E}\{\langle n, n'\rangle, \langle n', n\rangle\}$, *where* $\langle n, n'\rangle$ *denotes an operator with precondition* $Pre(o) = \{n\}$ *and add effect* $Add(o) = \{n'\}$ *and the cost of each operator is equal to the weight of the edge* $w(\langle n, n'\rangle)$ *with which it is associated*
- $G = T$

The planning problem P_S is equivalent to the ST problem S in the sense that every plan π for P_S encodes a CST C_π in S and vice versa. To see this, note that any plan π for P_S can be interpreted as the CST $C_\pi = \{\langle n, n'\rangle \mid \langle n, n'\rangle \in \pi\}$. Furthermore, by how the cost of each operator is obtained above, the weight of the resulting CST $w(C_\pi)$ and the cost of π are equal. It then follows that:

Theorem 4.3. *The Steiner trees for the problem $S = \langle\langle V, E\rangle, T\rangle$ are equivalent to the optimal plans for P_S and $c^*(S) = cost^*(P_S) = cost^*(P_S^+)$.*

As discussed in Section 2.4, the planning encoding of the ST problem is a problem with no deletes in which the preconditions of all actions have size 1, but the size of the goal set $|G|$ is unbounded. Since the problem has no delete effects, the value of the optimal delete relaxation heuristic h^+ captures the optimal cost of this encoding. However, the delete relaxation heuristics discussed until now that are based on the independence assumption do not: since all preconditions in the problem are unary, they are able to give optimal cost estimates for the cost of the path to each of the target nodes of the problem in isolation, but not for the set of goals T as a whole. Instead, the relaxed plans that they yield consist of the union of the shortest paths to each of the goals, which is known as a *trees of shortest paths* (Cormen et al., 2001).

The cost of the tree of shortest paths is not a good approximation to the cost of the ST problem. Yet many polynomial-time algorithms that compute better approximations exist (Charikar et al., 1998; Robins and Zelikovsky, 2000). We cannot build directly on these algorithms, however, as the graphs that underlie the delete relaxations of arbitrary planning problems are *directed hypergraphs*, rather than undirected graphs. However, as we will show below, it is possible to obtain better relaxed plans by borrowing some of the underlying ideas. We begin by developing an improvement procedure for CSTs in undirected graphs, and then show how to extend the procedure to directed graphs and later directed hypergraphs, which are equivalent to the delete relaxation problem.

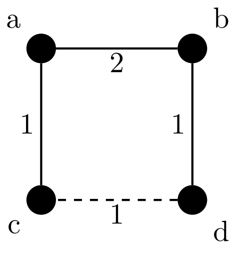
4.2 Improvement Algorithms for Steiner Trees

Given a CST C, the set of points spanned by C which are not in T are known as the *Steiner points* of C (Proemel and Steger, 2002). For a fixed set of Steiner points Q, it is easy to see that the lowest cost CST that can be constructed constitutes an MST over the subgraph of G induced by $Q \cup T$, as any tree with greater cost could be replaced with the MST over this set of points to yield a lower cost CST. The weight of the Steiner tree for a given problem S can then be stated as the minimum over all possible sets of Steiner points of the weight of the MST on the subgraph induced by the union of the set of terminal points and the set of Steiner points:

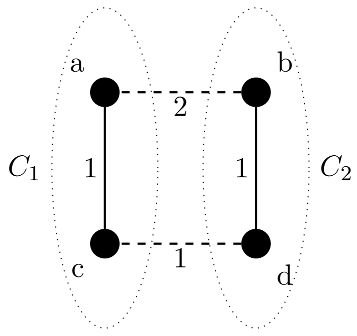
$$w^*(S) = min_{Q \subseteq (V \setminus T)} w(MST(G^{T \cup Q})) \qquad (4.1)$$

Solving the ST problem is then equivalent to finding the set of Steiner points Q which results in the induced subgraph $G^{T \cup Q}$ with the minimum cost MST. Though it is of course intractable to find this optimal set of Steiner points, the property suggests a fast test to eliminate a tree C spanning a set of nodes N, $T \subseteq N$, from consideration: if C is not an MST over the graph induced by N, C can be replaced in polynomial time by the MST that spans the same nodes with less cost. Furthermore, these improvements can be done incrementally with a simple algorithm that consists of removing an edge from C and checking whether the two connected components that result can be reconnected with a lower weight edge to give a new tree C' that spans the same set of nodes but has less weight (Figures 4.2 and 4.1). When no further edges can be replaced, it can be shown that C is an MST over the set of nodes that it spans.

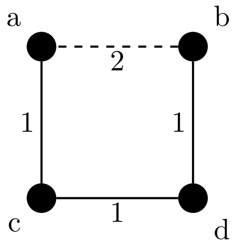
Taking into consideration that our overall goal is not to find an MST over the set of nodes spanned by C but rather to approximate the cost of the ST problem as closely as possible, this algorithm can be further improved. As the edge replacement algorithm replaces only single edges connecting two nodes in the candidate solution, the set of Steiner points Q of C is never modified. A version of the algorithm that also checks solutions with a modified set of Steiner points Q' is therefore guaranteed to give better results (recall Equation 4.1). Such an algorithm can be constructed by considering for replacement a *path* p in C whose internal nodes are all Steiner points, rather than only a single edge e (Figures 4.5 and 4.4). In other words, rather than replacing an edge e connecting two trees with a



(a) Suboptimal spanning tree with weight 4.



(b) Removal of edge $\langle a, b \rangle$ results in two connected components.



(c) Reconnecting C_1 and C_2 with edge $\langle c, d \rangle$ gives new MST with weight 3.

Figure 4.1: Edge replacement in undirected MSTs.

```
**Input**: A graph $G = \langle V, E \rangle$
**Input**: A tree $C$ in $G$
**Output**: A tree $C'$, with $w(C') \leq w(C)$

for $e \in C$ do
    $C_1, C_2 \leftarrow$ ConnectedComponents$(C \setminus e)$
    $E_r \leftarrow \{\langle n, n' \rangle \in E \mid n \in C_1 \wedge n' \in C_2\}$
    for $e' \in E_r$ do
        if $w(e') < w(e)$ then
            return $C_1 \cup C_2 \cup e'$
return $C$
```

Figure 4.2: Edge replacement algorithm for undirected graphs.

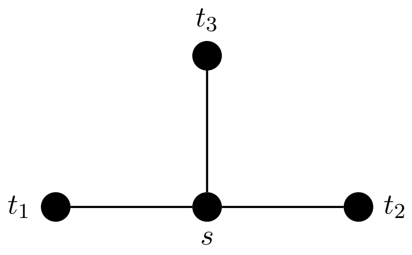
lower weight edge e' that connects the same two trees, the algorithm attempts to replace a path $p = \langle n_1, \ldots, n_k \rangle$ in C such that $n_i \notin T$ for $1 < i < k$. When removing a path p from C, care must be taken as more than two connected components C_i such that $C_i \cap T \neq \emptyset$ may result (Figure 4.3). For simplicity, we ignore such paths and consider only those whose removal results in exactly two connected components C_1 and C_2. If a new path connecting the two $p' = \langle n'_1, \ldots, n'_m \rangle$ with $n'_1 \in C_1$, $n'_m \in C_2$, $n'_2, \ldots, n'_{m-1} \notin C_1, C_2$, and $c(p') < c(p)$ can then be found, a new CST $C' = C_1 \cup C_2 \cup p'$ of lower cost results from replacing p with p'. The requirement that the internal nodes of the path not belong to either connected component ensures that the resulting structure contains no cycles and is a tree. In the undirected setting, this property could also be preserved by considering only minimal cost paths that reconnect the two components, however the criterion as stated here is more readily applicable to the directed graphs.

This more powerful improvement procedure generalizes the edge replacement algorithm, and may change the set of Steiner points Q of C.

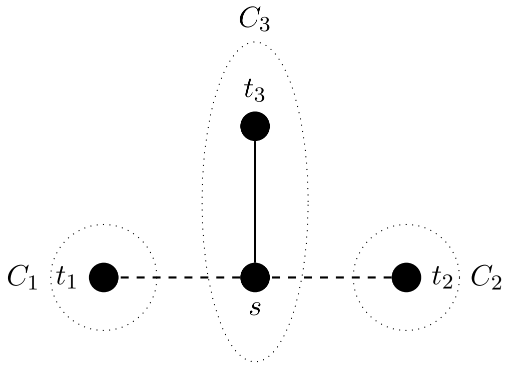
Improving Directed CSTs

The edge and path replacement algorithms discussed above can easily be extended to directed graphs. The one issue that arises is that while two undirected trees C_1, C_2 can be joined by a path from any node of C_1 to any node of C_2 to obtain a new tree that spans all the nodes of both C_1 and C_2, this is not the case for trees in directed graphs. When C_1 and C_2 are directed trees, adding a path that terminates in a node that already has an incoming edge will increase the in-degree of the node to 2, invalidating the tree property (see Figure 4.6). The paths or edges that are chosen in order to reconnect the two components must therefore end in nodes that have in-degree 0 in the individual connected components, or in other words the root nodes of each. Modifying the path improvement algorithm presented above to consider only such paths is a simple matter.

Furthermore, note that in directed (hyper)graphs representing



(a) Steiner tree for $T = \{t_1, t_2, t_3\}$.



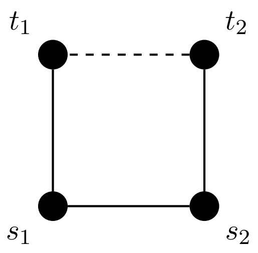
(b) Removal of path $p = \langle t_1, s, t_2 \rangle$ results in 3 connected components containing points in T.

Figure 4.3: Multiple connected components resulting from path removal.

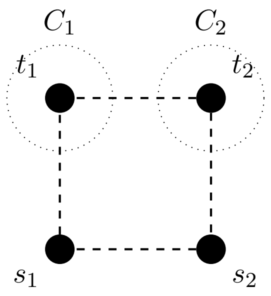
delete relaxation problems, solutions are necessarily rooted at the node i representing the initial state that results from applying the transformation implied by Proposition 2.12. Given such a CST, the removal of a path necessarily results in one component C^0 which is rooted at the initial state, and a second component C^+ which is not. In order to reconnect the two connected components, it is then sufficient to only consider paths from some node of C^0 to the root node of C^+.

4.3 Improving Relaxed Plans

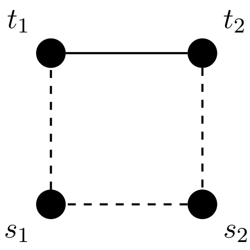
We now show how to adapt the algorithm described above to the setting of delete relaxation problems represented by hyper-



(a) Steiner tree with Steiner point set $Q = \{s_1, s_2\}$.



(b) Removal of path $p = \langle t_1, s_1, s_2, t_2 \rangle$ results in two connected components containing nodes in T.



(c) Reconnecting C_1 and C_2 with path $p = \langle t_1, t_2 \rangle$ gives new Steiner tree with Steiner point set $Q = \{\}$.

Figure 4.4: Path replacement in Steiner trees, $T = \{t_1, t_2\}$.

graphs, assuming without loss of generality that solutions take the form of relaxed plans π from a single initial fluent i (Proposition 2.12) to a single goal fluent g (Proposition 2.11), and that all actions have a single add effect (Proposition 2.10). We discuss below how to modify the procedure to handle multiple add effects.

We define three disjoint subsets of π: $\pi^+(y)$, $\pi^0(y)$, and $\pi^-(y)$ in terms of any fluent $y \in F(\pi)$, where $F(\pi)$ represents the set of fluents that appear in the precondition $Pre(o)$ of some operator $o \in \pi$. These subsets correspond respectively to the two connected components C^+ and C^0 and to the path p that

```
**Input**: An ST problem $S = \langle\langle V, E\rangle, T\rangle$
**Input**: A tree $C$ in $G$
**Output**: A tree $C'$, with $w(C') \leq w(C)$
$P_C \leftarrow \{p = \langle n_1, \dots, n_k\rangle \subseteq C \mid n_2, \dots, n_{k-1} \notin T\}$
**for** $p \in P_C$ **do**
    **if** $|\texttt{ConnectedComponents}(C \setminus p)| = 2$ **then**
        $C_1, C_2 \leftarrow \texttt{ConnectedComponents}(C \setminus p)$
        $P_r \leftarrow \{\langle n'_1, \dots, n'_m\rangle \in \langle V, E\rangle \mid n'_1 \in C_1 \wedge n'_m \in C_2\}$
        $P_r \leftarrow \{\langle n'_1, \dots, n'_m\rangle \in P_r \mid n'_2, \dots, n'_{m-1} \notin C_1, C_2\}$
        **for** $p' \in P_r$ **do**
            **if** $w(p') < w(p)$ **then**
                **return** $C_1 \cup C_2 \cup p'$
**return** $C$
```

Figure 4.5: Path replacement algorithm for undirected graphs.

connects them (Figure 4.7).

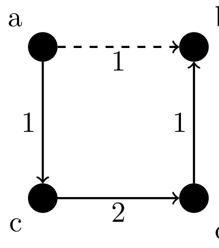
Given y, the structure analogous to the tree C^+ rooted at y in the directed tree setting is the portion of the plan $\pi^+(y)$ that is *dependent* on y, defined recursively as follows:

1. $o \in \pi$ is dependent on y if $y \in Pre(o)$.

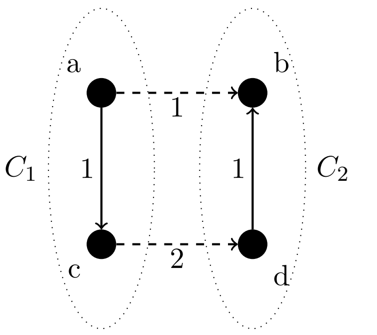
2. $o \in \pi$ is dependent on y if it is dependent on some fluent x added by an operator that is dependent on y.

The algorithm shown in Figure 4.8 computes $\pi^+(y)$ incrementally, iterating over the set of operators in π and terminating when no further actions can be added to the set. This computation can also be implemented recursively with an algorithm similar to relaxed plan extraction.

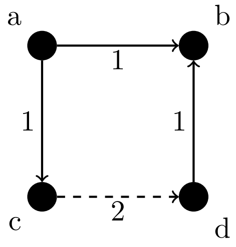
The portion of the relaxed plan that corresponds to the path p between the two directed trees is termed $\pi^-(y)$, and is the set of actions in π that are required *only* in order to achieve the fluent y, i.e. that would not be necessary if y were already present in the initial state from which π is calculated. Note that



(a) Optimal directed MST with weight 4.



(b) Removal of edge $\langle c, d\rangle$ results in two connected components.



(c) Reconnecting C_1 and C_2 with edge $\langle a, b\rangle$ would invalidate tree property.

Figure 4.6: Preserving the tree property in directed graphs.

this set of actions is distinct from the set of actions in π that form a plan for y: an action may appear in the portion of the plan that achieves y and also in the plan for the preconditions of some action that is not dependent on y. In Figure 4.9a, for example, the operator o_p is part of the plan for y yet does not belong to $\pi^-(y)$. This consideration is related to the discussion above of the removal of paths possibly resulting in more than two connected components (Figure 4.3). $\pi^-(y)$ can be computed by first extracting a relaxed plan $\pi(g \mid s \cup \{y\})$ for the problem that "assumes" y by returning the empty plan whenever y is encountered as a precondition, and then removing these actions from the global plan π in order to obtain the actions that are required exclusively in order to achieve y:

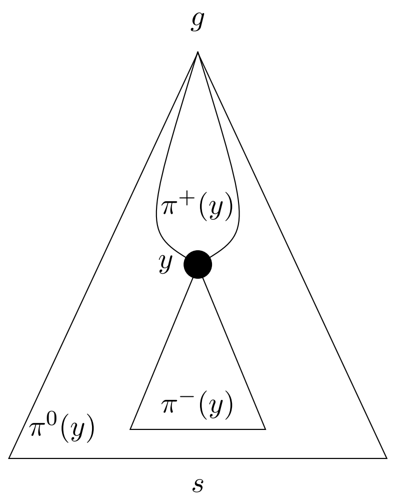
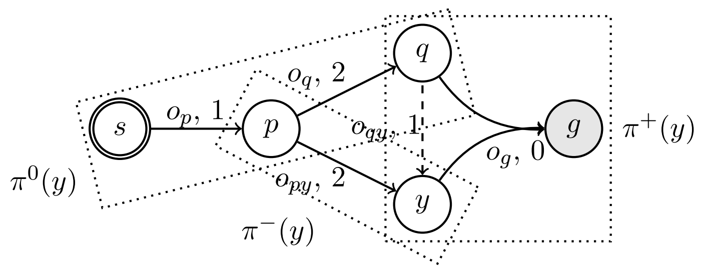


Figure 4.7: Visual representation of plan components.

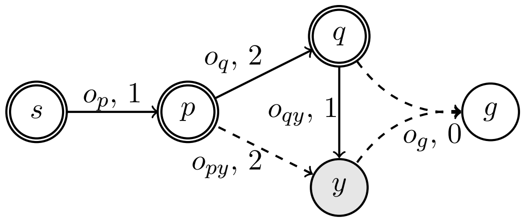
Input: A delete-free planning problem $P = \langle F, \{i\}, O, G \rangle$
Input: A plan π in P from i to g
Input: A fluent y
Output: The set of operators constituting $\pi^+(y)$ with regards to goal g

```
$\pi^+(y) \leftarrow \emptyset$
$dep \leftarrow \{y\}$
$fixpoint \leftarrow \mathsf{false}$
while $!fixpoint$ do
    $fixpoint \leftarrow \mathsf{true}$
    for $o \in \pi, o \notin \pi^+(y)$ do
        if $dep \cap Pre(o) \neq \emptyset$ then
            $\pi^+(y) \leftarrow \pi^+(y) \cup \{o\}$
            $dep \leftarrow dep \cup \{Add(o)\}$
            $fixpoint \leftarrow \mathsf{false}$
```

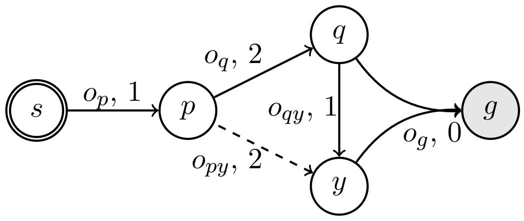
Figure 4.8: Algorithm for computing $\pi^+(y)$



(a) A delete relaxation plan with cost 5, $\pi^+(y) = \{o_g\}$, $\pi^-(y) = \{o_{py}\}$, $\pi^0(y) = \{o_p, o_q\}$



(b) Plan for y from augmented initial state $\{s, p, q\}$ is $\{o_{qy}\}$, with cost 1



(c) Replacing $\pi^-(y)$ with $\pi(y|s')$ yields relaxed plan with cost 4

Figure 4.9: Partial plan replacement in relaxed plans.

$$\pi^-(y) = \pi \setminus \pi(g \mid s \cup \{y\}) \tag{4.2}$$

Finally, the part of the relaxed plan corresponding to the connected component rooted at the initial state fluent i can be computed by simply subtracting the two sets above from the complete plan π:

$$\pi^0(y) = \pi \setminus (\pi^+(y) \cup \pi^-(y)) \tag{4.3}$$

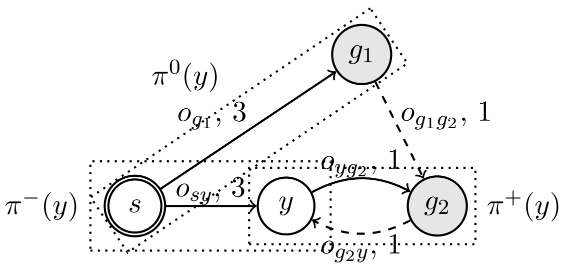
With these three disjoint sets defined, it is now possible to lay out the algorithm that performs partial plan substitution in relaxed plans. The algorithm follows the same intuition as that of the algorithms given above, yet here the place of the computation of a path from any one of the nodes of a connected component to the root of the other is taken by the computation of a relaxed plan $\pi(y; s_y)$ for y from an *augmented state* s_y that is obtained by adding the fluents achieved by the plan component $\pi^0(y)$ to $\{i\}$.

In the path improvement algorithm, we prevented the formation of cycles when reconnecting the two connected components C_1 and C_2 by only considering paths whose internal nodes did not belong to either. Here the same effect is achieved by computing the new relaxed plan $\pi(y; s_y)$, in a modified problem Π' in which fluents added by an operator in $\pi^+(y)$ are removed from the problem. Figure 4.10 shows how cycles can result when these actions are considered in the recomputation of the plan for y. No assumptions about how this relaxed plan is computed are made, and any existing technique, for example those discussed in Chapter 3, can be used. A single iteration of this procedure that checks whether an improved partial relaxed plan can be found for any $y \in F(\pi)$ is shown in Figure 4.11. This procedure is repeatedly called until the plan that is returned is equal to the plan given as input, indicating that no further improvements can be performed.

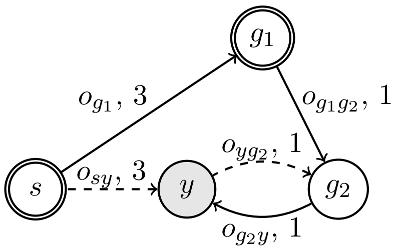
Implementation

The implementation of our improvement procedure differs from the algorithm discussed above in certain respects. First of all, as explicitly manipulating sets representing relaxed plans is expensive, we instead keep a table that lists for each fluent f in the problem a *best supporter* $bs(f)$. When a cheaper plan achieving y is found, it is then sufficient to change the entry for y and the entries for the set of fluents $F(\pi(y; s_y)$. The previously discussed relaxed plan extraction procedure (Figure 3.3) will then extract the modified plan.

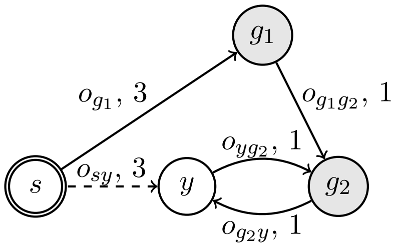
This method of manipulating best supporters rather than sets is



(a) Initial delete relaxation plan.



(b) Plan for y from initial state $s_y = \{s, g_2\}$ when no fluents are excluded. $\pi(y; s_y) = \{o_{g_1g_2}, o_{g_2y}\}$.



(c) Replacement of $\pi^-(y)$ with plan obtained for y without exclusions leads to a cycle.

Figure 4.10: Exclusion of actions in partial plan recomputation.

```
Input: A delete-free planning problem
       $P = \langle F, \{i\}, O, \{g\} \rangle$
Input: A plan $\pi$ in $P$
Output: A plan $\pi'$ in $P$ such that $cost(\pi') \leq cost(\pi)$
for $y \in F(\pi)$ do
    $\pi^+ \leftarrow$ Compute-$\pi^+(y)$
    $\pi^- \leftarrow$ Compute-$\pi^-(y)$
    $\pi^0 \leftarrow$ Compute-$\pi^0(y)$
    $s_y \leftarrow \{i\} \cup \bigcup_{o \in \pi^0} Add(o)$
    $exc_y \leftarrow \bigcup_{o \in \pi^+} Add(o)$
    $\pi(y; s_y) \leftarrow$
    ComputeRelaxedPlanExcluding($s_y, \{y\}, exc_y$)
    if $cost(\pi(y; s_y)) \leq cost(\pi^-)$ then
        return $\pi^+ \cup \pi^0 \cup \pi(y; s_y)$
return $\pi$
```

Figure 4.11: Local Steiner tree procedure for improving relaxed plan π.

also relevant for the assumption stated above that each action has a single add effect. When this is assumed, an action appearing in a plan is necessarily the best supporter for the single fluent that it adds. However, applying the transformation required to enforce this property is costly and unnecessarily increases the size of the problem. To avoid this transformation, the exclusion criteria discussed above has to be modified. To obtain the new problem Π' in which $\pi(y; s_y)$ is to be computed, it is sufficient to remove from Π all fluents whose *best supporters* are in the set $\pi^+(y)$. The set *exc* in the algorithm shown in Figure 4.11 can then be obtained as

$$exc_y \leftarrow \{p \mid a_p \in \pi^+(y)\}$$

The main drawback to the use of this improvement procedure in all settings is the cost of its computation, which results primarily from the repeated calls to the heuristic used to obtain a relaxed plan for each fluent $y \in F(\pi)$ from the augmented state s_y. Two possible strategies could ameliorate this overhead. One of these is to decrease the number of calls made to obtain relaxed plans; this can be done for example by attempting the improvement procedure on only a subset of the fluents in $F(\pi)$, such as the set of goals of the problem or the set of delete-relaxation landmarks. The second is to decrease the time taken by each call to the heuristic. This can be done by observing that the initial states s_y used in the computation of new relaxed plans are similar to s, and indeed since $s \subseteq s_y$ for all y, the costs of all fluents in the problem as computed by heuristics such as h^{add} and $h^{\max}$ are guaranteed to decrease compared to their costs in s. The heuristic values computed initially for s can then be used as seed values for the new computation, resulting in fewer iterations before the values converge (Liu et al., 2002).

4.4 Experimental Results

In order to assess the impact of the relaxed plan improvement procedure we have described, we compare the performance of a planner using an independence assumption based delete relaxation heuristic to the performance of the same planner using as a heuristic value the cost of the relaxed plan resulting from the improvement procedure. The independence assumption based heuristic is the additive heuristic h^{add}, which offers a good tradeoff between coverage and quality (see Section 3.5 for further information). As in the evaluation of previous heuristics, the planner uses a greedy best first search algorithm with delayed evaluation and a second open list for nodes resulting from helpful actions, and the set of test domains is the set of satisficing instances from the most recent International Planning Competition (IPC6). All experiments discussed below were run on Xeon Woodcrest computers with clock speeds of 2.33 GHz, using a 2GB memory limit and a time cutoff of 1800 seconds.

In our evaluation of independence assumption based heuristics, we discuss three configurations for the heuristics: (a) using costs as stated in the problem, (b) using an added *base cost* for zero cost operators, and (c), using the size of the relaxed plan as the heuristic estimate. Here we report only experiments performed

Domain	(a)	(b)	(c)	(d)	(e)
cybersec (30)	0.99	13	+2/-4	+1/-1	+0/-0
elevators (30)	0.79	86	+12/-0	+0/-10	+1/-9
openstacks (30)	1.00	3	+0/-0	+0/-0	+0/-0
parcprinter (30)	0.81	3	+9/-1	+3/-10	+0/-0
pegsol (30)	0.64	6	+0/-1	+15/-13	+7/-12
scanalyzer (30)	0.92	2	+2/-1	+4/-13	+5/-7
sokoban (30)	0.93	6	+0/-4	+10/-13	+11/-4
transport (30)	0.85	46	+1/-0	+4/-8	+5/-6
woodworking (30)	0.90	11	+1/-2	+6/-18	+5/-17
total/avg. (270)	0.81	7	+27/-13	+43/-86	+34/-55

Table 4.1: Evaluation of Steiner improvement procedure. Column (a): average ratio of the cost of the relaxed plan after improvement to before improvement, (b): average slowdown in number of heuristic estimates per second, (c): number of problems solved using the improvement procedure but not without, and vice versa, (d): number of problems in which more/less nodes were evaluated to find a solution, (e): number of problems for which more costly/less costly plans were found.

with the second configuration, which offers the best tradeoff between coverage and plan quality.

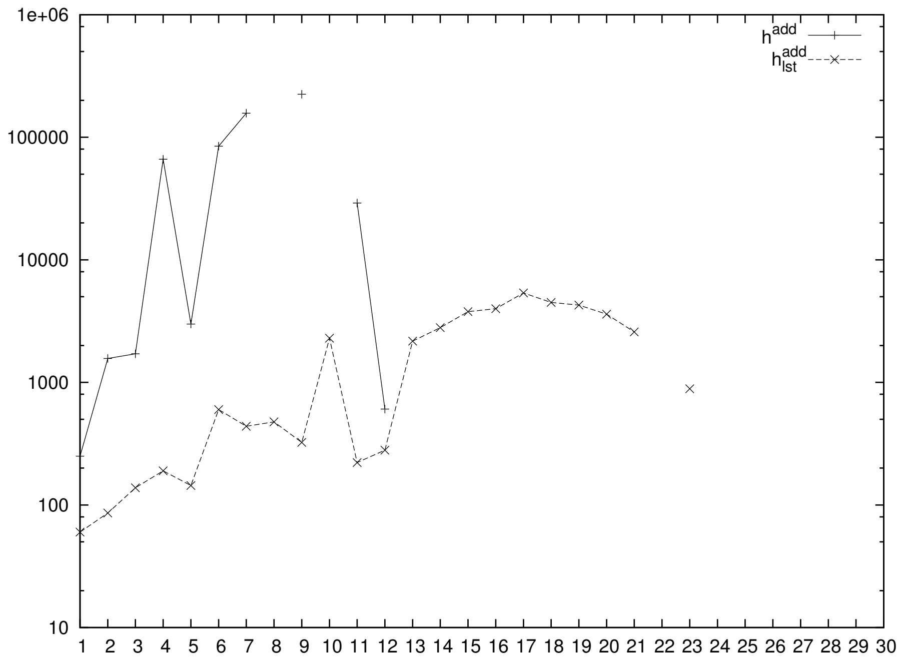
In Table 4.1, the columns compare the behaviour of the local Steiner tree improvement procedure (LST) to the bare additive heuristic in different respects. In the first column, the average ratio of the cost of the relaxed plan resulting from applying the LST procedure to the cost of the original plan is shown. In almost all domains, the cost of the relaxed plan is improved, with the openstacks domain being the only exception. The other domains benefit to various degrees, with the best case being the pegsol domain, in which plans obtained with the LST procedure have only 64% of the cost of the original relaxed plans. We note also that since the values shown are averages for all of the evaluated states, those in which the plan cannot be improved significantly affect the result. In many states, the improvement to the relaxed plan is significantly larger than the average.

In column (b), we show the approximate average slowdown in the number of heuristic evaluations per second that results from the using the LST procedure. As expected, the many heuristic evaluations required to compute the result of the LST in a single state take their toll, and evaluation is significantly slower in all domains. The worst cases occur in the elevators and transport instances, in which heuristic evaluation is 86 and 46 times slower, respectively. The average slowdown across all domains, weighted by the number of states evaluated in each, is 7.

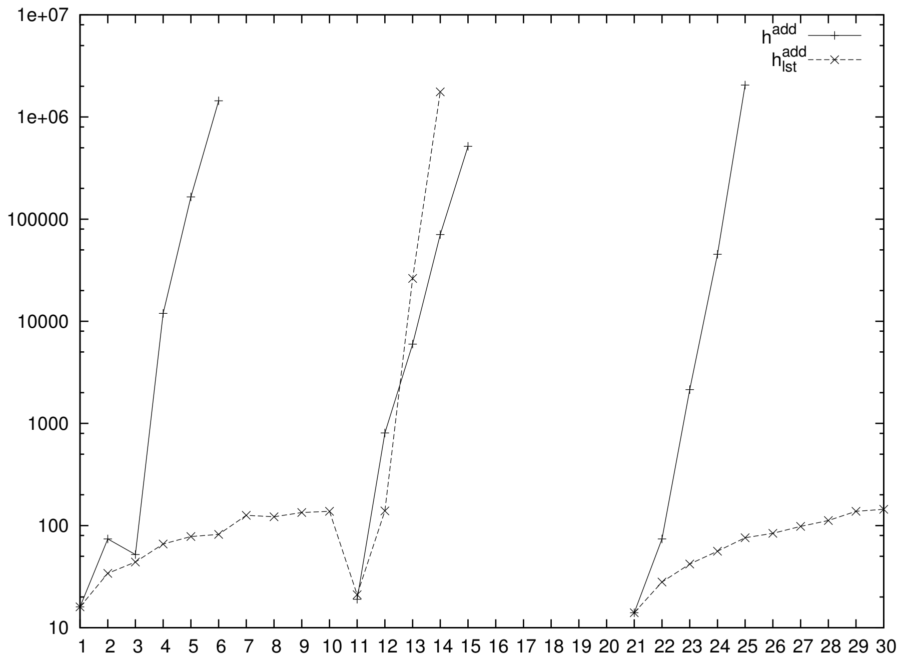
Column (c) shows the differences in the number of problems solved when the heuristic is compared to the bare additive heuristic. Somewhat surprisingly, the large slowdowns reported in column (b) are not hugely detrimental to coverage: in fact, use of the procedure results in the planner solving an additional 14 problems across all domains. This improvement is concentrated in the elevators and parcprinter domains, in which the planner solves 12 and 8 more problems respectively. In domains for which the procedure does not increase informativeness, the effect of the slowdown is to slightly decrease the coverage of the planner. The worst case for this occurs in the sokoban domain, in which 4 problems fewer are solved.

The change in the number of nodes expanded with the procedure, considering only the problems solved with both planners, is shown in column (d). The improved relaxed plans are significantly more informative in the two domains that were also responsible for the increase in coverage, elevators and parcprinter. In fact, in these two domains the number of nodes that have to be evaluated decreases by orders of magnitude when the improvement procedure is used (Figure 4.12). Some improvement is also observed in the woodworking and scanalyzer domains. In the other domains, the procedure does not affect informativeness. Overall, the heuristic results in fewer heuristic evaluations in 86 problems, and more in only 43.

Finally, column (e) shows the results considered from the point of view of plan cost. The LST procedure results in the generation of lower cost plans in the elevators and woodworking domains, and to a lesser extent in the pegsol domain. The increased coverage in the parcprinter domain does not come with an accompanying decrease in plan quality. In the sokoban domain, which is in



(a) The elevators domain.



(b) The parcprinter domain.

Figure 4.12: Node expansions with the Steiner improvement procedure.

general resistant to improvements in delete relaxation estimates, plan costs significantly increase with the use of the procedure. Globally, the LST heuristic results in 55 plans solved with lower cost and 34 with higher cost among those problems solved with both heuristics.

4.5 Conclusions

We have discussed the relationship between the well-known Steiner tree problem and the delete relaxation, showing that in the Steiner tree problem the independence assumption results in a tree-of-shortest-paths approximation. We have taken advantage of a simple property of Steiner trees to formulate an improvement procedure for the undirected Steiner tree problem and shown how it can be modified to work for the directed version, and in turn, for the directed hyperpath problem which is equivalent to the delete relaxation problem. This procedure constitutes the first approach that allows the computation of suboptimal relaxed plans that do not depend on the independence assumption in their estimates. The increased computational effort required to compute these relaxed plans turns out to pay off in several settings, and the use of the procedure increases both coverage and plan quality in the domains studied.

Part III

Beyond The Delete Relaxation

Chapter 5

Landmarks for the Delete Relaxation and Beyond

Landmarks are necessary properties of solutions to planning problems. These may be formulas over the set of fluents, implying that the formula must be made true in some state during the execution of any valid plan, or formulas over operators, interpreted as statements about operators that must necessarily belong to plans. In this chapter we give the equations that describe the complete set of single fluent and operator landmarks for the delete relaxation, and show how to apply our method to a recently proposed transformation of the planning problem, allowing the discovery of conjunctive landmarks and landmarks beyond the delete relaxation.

5.1 Introduction

Landmarks in a planning problem are necessary features of solutions to planning problems. Fluent landmarks are formulas over the set of fluents of the problem that must be satisfied by some state that occurs during the execution of any valid plan:

Definition 5.1 (Fluent landmarks). *A fluent landmark L is a formula over the set of fluents F of a planning problem, such* *that any valid plan* $\pi = \langle o_1, \ldots, o_n \rangle$ *has a prefix* $\pi' = \langle o_1, \ldots, o_i \rangle$*, possibly of length* 0*, whose application in the initial state results in a state in which* L *is true, i.e.* $s_0[\pi'] \models L$*.*

It can be seen by choosing the empty prefix $|\pi'| = 0$ that all formulas true in the initial state of the problem are also landmarks. Similarly, choosing the prefix to be the entire plan π implies that all formulas entailed by the goal are landmarks for the problem as well.

Landmarks consisting of a single disjunction or conjunction are referred to as *disjunctive landmarks* and *conjunctive landmarks*, respectively. *Orderings* over fluent landmarks are statements about the order in which they must be made true. While more complicated ordering criteria have been proposed (Hoffmann et al., 2004), here we limit the discussion to two types:

Definition 5.2 (Natural ordering)**.** *Given two landmarks* L_1 *and* L_2*, there is a* natural ordering $L_1 \prec_{\mathrm{n}} L_2$ *if for any plan* $\pi = o_1, \ldots, o_n$*,* $s[o_1, \ldots, o_j] \models L_2$ *implies that there exists* $i < j$ *such that* $s[o_1, \ldots, o_i] \models L_1$*.*

Definition 5.3 (Greedy necessary ordering)**.** *Given two landmarks* L_1 *and* L_2*, there is a* greedy-necessary *ordering* $L_1 \prec_{\mathrm{gn}} L_2$ *if for any plan* $\pi = o_1, \ldots, o_n$*,* $s[o_1, \ldots, o_j] \models L_2$ *and* $s[o_1, \ldots, o_i] \not\models L_2$ *for all* $i < j$ *implies* $s[o_1, \ldots, o_{j-1}] \models L_1$*.*

Intuitively, a natural ordering $L_1 \prec_{\mathrm{n}} L_2$ states that L_2 cannot be made true without making L_1 true first, and a greedy natural ordering $L_1 \prec_{\mathrm{gn}} L_2$ states that L_1 must be true immediately before L_2 is made true for the *first* time.

Definition 5.4 (Operator landmarks)**.** *An operator landmark* L *is a formula over the set of operators* O *of a planning problem, such that when any valid plan* π *is interpreted as a truth assignment to the set of operators in the problem, with those operators appearing in* π *having the value* true *and those not appearing in* π *having the value false,* L *is satisfied.*

The problem of deciding whether a given formula is a landmark for a planning problem is the `Landmark` problem:

Definition 5.5 (Landmark). *Given a planning problem Π, and a formula L over the set of fluents F or the set of operators O, $\texttt{Landmark}(\Pi, L)$ is the following decision problem:*

INSTANCE: *A planning problem Π and a landmark formula L.*

QUESTION: *Is L a landmark for Π?*

The Landmark problem is known to be PSPACE-complete (Hoffmann et al., 2004) even when formulas are restricted to size 1, e.g. single fluent and single action landmarks. Approaches to landmark finding therefore focus on finding landmarks for the delete relaxation Π^+ of planning problems, in which setting the problem of deciding $\texttt{Landmark}(\Pi^+, L)$ when L is restricted to single fluent and operator landmarks can be shown to be in P (Hoffmann et al., 2004). Finding the complete set of single fluent and operator landmarks for Π^+ is therefore also in P, as every fluent and operator can be tested with this polynomial procedure. This problem can also be stated in terms of hypergraphs or AND/OR graphs, in the case of hypergraphs as the problem of finding all edges and nodes included in every directed hyperpath for a set of initial and goal nodes, and in the case of AND/OR graphs as the problem of finding all AND and OR nodes v such that $v \in V^J$ for any justification J for a set of initial and goal nodes.

The naive approach referred to above of simply checking every fluent and operator in the problem to determine whether it is a landmark is too costly in practice. Most previously proposed methods have instead focused on variations of a technique known as *backchaining*. Backchaining methods start from a known single fluent landmark g, such as a goal fluent, and attempt to find more landmarks by reasoning backwards from g (Hoffmann et al., 2004). This can be done by checking whether the set of operators $O_g = \{o \mid g \in Add(o)\}$ adding g share a common precondition p, and if this is the case, adding p to the set of known landmarks. Backchaining can then be applied from p. This relies on the fact that achieving g, which is known to be a landmark, requires the application of some operator $o \in O_g$, all of which require p as a precondition. This method typically

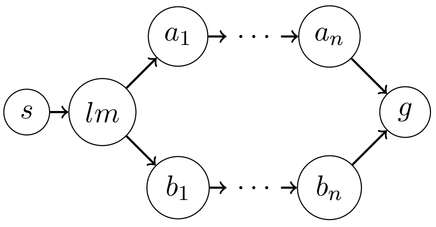


Figure 5.1: n levels of lookback are required to detect that lm is a landmark for g.

fails to find many landmarks, and is therefore enhanced with a *lookahead* procedure, which consists of choosing from each precondition set $pre(o)$ for $o \in O_g$ a single fluent p_o, and creating a disjunctive formula L over this set. Since some action in O_g necessarily must be applied, and its preconditions made true in order to achieve g, L is also a landmark for Π^+. To find further single fluent landmarks, $L = L_1 \vee \ldots \vee L_n$ can then be backchained from by checking whether all operators adding any one of the disjuncts L_i share a precondition, or using the disjunctive landmark itself as a landmark (Richter et al., 2008). Note however, that each time backchaining occurs from a disjunctive landmark, the number of disjunctions that must be considered grows exponentially in the number of actions adding any one of the disjuncts. In order to make the problem tractable, a limit is therefore imposed on the number of times backchaining occurs from consecutive disjunctive landmarks (known as the *lookahead depth*), and the size and type of admitted disjunctive landmarks. Due to these limits, backchaining is incomplete and does not find all of the landmarks of the delete relaxation. Furthermore, it is easy to construct a problem in which a lookahead depth of n is required to detect that a fluent is a landmark (Figure 5.1).

5.2 Optimal Π^+ Landmarks

Though the methods that are currently most in use rely on the incomplete method of backchaining, one algorithm has been proposed that instead of reasoning backwards from the goals of the problem by applying the backchaining method recursively, computes landmarks through *forward* propagation in a relaxed planning graph (RPG). This algorithm is sound and complete according to the simple and intuitive criterion of *causality*, which excludes "incidentally" achieved facts that are added by some action in the plan, but are neither used as preconditions by some other action nor are goals (Zhu and Givan, 2003):

Definition 5.6 (Causal landmarks)**.** *A fluent landmark L is a* causal landmark *for a problem* Π *if it is either satisfied by the set of goal fluents of* Π*,* $G \models L$*, or if for all valid plans* π *for* Π*,* $Pre(o) \models L$ *for some* $o \in \pi$*.*

The algorithm works by associating with each action or fluent node at every level of the RPG a *label* consisting of the set of facts that must be made true in order to reach it. In the first level of the RPG, each initial state fact is associated with a label containing only itself. The labels of the nodes appearing at following levels are obtained by combining the labels of the nodes in previous layers in two different ways:

- The label for an operator node o at level i is the union of the labels of all its preconditions at level $i-1$.
- The label for a fluent node f at level i is the intersection of the labels of all operator nodes adding it at level $i-1$ (possibly including no-op actions), plus the fact itself.

Intuitively, these rules state that for a fluent f to be a landmark for an operator o, it is sufficient that f be a landmark for *some* precondition of o, and that for a fluent f to be a landmark for another fluent f' at a given level, either $f = f'$ or f must be a landmark for *all* operators that achieve f' at that level.

Given these propagation rules, the label associated with a fluent or operator node at any level i is a *superset* of the set of causal landmarks for this fluent or operator in Π^+. If the criteria above are applied until a fixpoint is reached, *i.e.* until no further changes occur in the node labels from layer to layer, the labels for the goal nodes in the last layer are exactly the causal landmarks for Π^+.

Our contribution is to observe that these labels can be characterized as the unique maximal solution to the following set of equations, where maximal is interpreted in terms of set inclusion:

$$
\begin{aligned}
LM(G;s) &\stackrel{\text{def}}{=} \bigcup_{g\in G} LM(g) \\
LM(p;s) &\stackrel{\text{def}}{=} \begin{cases} \{p\} & \text{if } p \in s \\ \{p\} \cup \bigcap_{\{o \mid p\in Add(o)\}} LM(o;s) & \text{otherwise} \end{cases} \\
LM(o;s) &\stackrel{\text{def}}{=} \{o\} \cup \bigcup_{p\in Pre(o)} LM(p)
\end{aligned}
$$

Theorem 5.7. *For any planning problem Π^+ with no deletes, the system of equations $LM(\cdot)$ has a unique* maximal *solution, where maximal is defined with regard to set inclusion, and this solution satisfies*

$$p \in LM(q) \iff p \textit{ is a causal landmark of size } 1 \textit{ for } \{q\} \textit{ in } \Pi^+$$

for $p \in F, O$. Moreover, for any node set G, $LM(G)$ is the set of causal landmarks of size 1 for G in Π^+.

Proof. Let $LM_{\mathrm{c}}(p)$ denote the complete set of causal landmarks for p. A solution to the system of equations exists, as it is satisfied by setting $LM(p) = LM_{\mathrm{c}}(p)$ for all p. To show that LM_{c} is the unique maximal solution, we show that all solutions to $LM(\cdot)$ satisfy $p \in LM(q) \Rightarrow p \in LM_{\mathrm{c}}(q)$. Let a counterexample X to this implication be a tuple $\langle p, q, \pi\rangle$, where p and q are operators or fluents, such that

- π is a plan that adds q, if q is a fluent, or $q \in \pi$, if q is an operator,
- $p \neq q$,
- $p \in LM(q)$, and
- $p \notin \bigcup_{o\in\pi} Pre(o)$, if p is a fluent, or $p \notin \pi$, if p is an operator.

Assume that such a counterexample exists and choose one with *minimum size*, where we define the size of a counterexample to be $|X| = |\pi|$. We consider two cases:

$q \in F$ Let o' be the operator in π that adds q. Since $p \neq q$, $p \in \bigcap_{\{o|q \in Add(o)\}} LM(o)$, and $p \in LM(o')$. Since $p \notin \pi$, $p \neq o'$. Let r be one of the preconditions of o' such that $p \in LM(r)$. Such a precondition must exist since $p \in \bigcup_{r \in Pre(o')} LM(r)$. Then there exists $\pi_r \subseteq \pi$ that constitutes a plan for r, and $|\pi_r| < |\pi|$ since $o' \notin \pi_r$. We then have that $p \in LM(r)$ and $p \notin (\pi_r \cup \bigcup_{o \in \pi_r} Pre(o))$. $X' = \langle p, r, \pi_r \rangle$ is therefore a smaller counterexample, contradicting our choice of X to have minimum size. Such a counterexample therefore cannot exist.

$q \in O$ Since $p \neq q$, $p \in \bigcup_{r \in Pre(q)} LM(r)$. Then there exists $r \in Pre(q)$ such that $p \in LM(r)$. Let $\pi_r \subseteq \pi$ be the minimal subset of π that constitutes a plan for r. We have that $|\pi_r| < |\pi|$, since $q \notin \pi_r$. Then as before $X' = \langle p, r, \pi_r \rangle$ is a counterexample with $|X'| < |X|$, and X cannot exist.

□

The unique maximal solution to the $LM(\cdot)$ equations can be found in polynomial time with the generalized versions of the Bellman-Ford procedure (Liu et al., 2002), in the same way that this algorithm can be adapted to compute the values of similar declaratively defined heuristics such as those discussed in Chapter 3. One way to compute the solution is to initialize the set of landmarks for the initial state fluents $p \in s$ to $\{p\}$, and the set of landmarks for all other fluents and operators in the problem to $O \cup F$, and then iteratively refine these sets by interpreting the equations as update rules. If the updates are performed according to the order in which nodes are generated in the relaxed planning graph (*i.e.*, all nodes in the first layer, then all nodes in the second layer, etc.), then the RPG label propagation algorithm by Zhu & Givan is obtained. If only fluent landmarks are sought, the equation for operators can be modified to not include the operator itself, $\{o\}$, in $LM(o)$.

Orderings between landmarks of the types discussed above can easily be inferred and recorded during the computation of the landmarks. A natural ordering $p \prec_{\mathrm{n}} q$ exists when $p \in \mathit{LM}(q)$, since it implies that q can not be achieved without achieving p first. Greedy necessary orderings between two facts $p \prec_{\mathrm{gn}} q$ can be inferred by maintaining for each fact q a set of operators that are its *first achievers*, $\mathit{FA}(q) \overset{\mathrm{def}}{=} \{o \mid q \in \mathit{Add}(o) \wedge q \notin \mathit{LM}(o)\}$. This is the set of operators that add q and do not have q as a landmark, implying that they can be used to achieve q for the first time. A greedy necessary ordering $p \prec_{\mathrm{gn}} q$ then exists when $p \in \bigcap_{o \in \mathit{FA}(q)} \mathit{Pre}(o)$. These orderings can be discovered and recorded during the computation of the landmarks and do not require any additional post-processing step.

5.3 Landmarks for the Π^m Problem

The method above improves on backchaining by guaranteeing that the set of landmarks found for the delete relaxation is complete with regard to a well-defined criterion, and on the Zhu & Givan algorithm by giving a declarative definition of the landmarks that it finds, thus eliminating the need for the relaxed planning graph and allowing other computational techniques to be applied. However it is possible to further extend landmark discovery techniques by taking advantage of a recently proposed compilation that converts a planning problem into a problem without deletes, yet whose fluents and operators encode information about the deletes present in the original problem.

The $h^{\max}$ heuristic discussed in Chapter 3 is also known as the h^1 heuristic, as it can be seen as recursively estimating the cost of a set of fluents as the cost of the *single* most expensive fluent in the set. Generalizing this heuristic with a parameter m results in the h^m family of heuristics, which as the name implies estimate the cost of a set as the cost of the most expensive subset of the set of size less than or equal to m (Haslum and Geffner, 2000). The Π^m compilation encodes the relaxation implied by the h^m heuristics in a problem with no delete effects, whose fluents correspond to sets of fluents of size less than or equal to m in Π, and whose operators are obtained by making explicit in the precondition and add effects of the original operators of the problem those facts which, while not required or added by an action, may occur in the state in which the operator is applied and persist after its application, allowing them to be achieved in conjunction with the effects of the action (Haslum, 2009). This is done by creating for each operator o in Π a *set* of operators in the new problem, each having as a precondition in addition to the precondition of o itself, a set of facts C of size at most $m-1$ such that C is disjoint from $Add(o)$ and $Del(o)$. For a set C and action o, the operator o_C is then given by:

$$\begin{aligned} Pre(o_C) &= \{S \mid S \subseteq (Pre(o) \cup C) \wedge |S| \leq m\} \\ Add(o_C) &= \{S \mid S \subseteq (Add(o) \cup C) \wedge |S| \leq m\} \\ Del(o_C) &= \emptyset \end{aligned}$$

An operator landmark o_C found for Π^m then corresponds to an operator landmark o in Π, and a fluent landmark S in Π^m representing the set of fluents $\{p_1, \ldots, p_m\}$ in Π corresponds to a conjunctive landmark $L = p_1 \wedge \ldots \wedge p_m$ in Π. These landmarks take into account delete information in the original problem Π, allowing landmarks that are not necessarily landmarks of the delete relaxation to be found for the first time.

Any landmark finding technique applicable to the delete relaxation is also applicable to the problem with no deletes resulting from the Π^m compilation. However the completeness property of the method described above guarantees that as the value of m goes to infinity, the set of landmarks found for Π approaches the complete set of conjunctive landmarks of the problem, just as the values of the h^m heuristic approach h^*.

Example

Consider the blocksworld problem of Figure 5.2. Apart from trivial landmarks such as those facts belonging to the initial state or goal, the complete set of causal delete-relaxation landmarks and orderings is *clear B* $\prec_{\text{gn}}$ *holding B*, implying that *holding B* must be made true in some state by any valid plan, and that

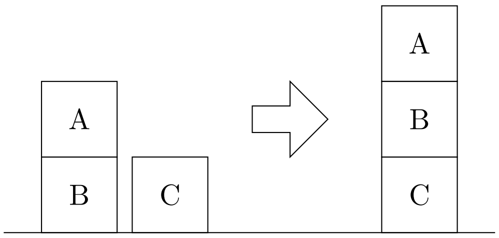


Figure 5.2: An instance of the blocksworld domain.

clear B must be true in the state that immediately precedes it. In contrast, when the landmarks computation is applied to the Π^2 compilation of the problem, one of the obtained chains of orderings is the following:

$$(\mathit{clear}\ B \land \mathit{holding}\ A) \prec_{\text{gn}} (\mathit{clear}\ B \land \mathit{on\text{-}table}\ A) \prec_{\text{gn}} (\mathit{holding}\ B \land \mathit{on\text{-}table}\ A) \prec_{\text{gn}} (\mathit{on}\ B\ C \land \mathit{on\text{-}table}\ A) \prec_{\text{gn}} (\mathit{on}\ B\ C \land \mathit{holding}\ A)$$

where $a \land b$ is a conjunctive landmark that implies that a and b must be true simultaneously in some state. These landmarks and orderings are only a subset of those found by the procedure, yet provide an almost complete roadmap for solving the problem.

The additional landmarks found in this way are not only conjunctive: the consideration of delete effects may also result in the discovery of fact landmarks for Π that are not landmarks in the Π^+ problem. In this example, the facts *holding A* and *ontable A* are also implied to be landmarks, as they are part of a conjunctive landmark.

Implementation

In the implementation of the Π^m compilation, we eliminate some unnecessary fluents by discarding fluent sets in which a subset of the fluents are *mutex* with one another, implying that they cannot occur together. Since these fluents cannot be achieved together in the original problem, they can not constitute landmarks for achievable fluents either.

Landmark sets in the computation are represented as ordered lists, which allow both the union and the set intersection operations to be performed in time linear in the size of the sets.

5.4 Experimental Results

We implemented the Π^m transformation and the computation of landmarks as discussed in Section 5.3. Here, we try to answer three main questions: whether our approach finds landmarks not found by previous approaches, whether these landmarks contain interesting information, and finally, whether current planners can exploit this information. All experiments were run on 2.3 GHz AMD Opteron machines using a 2 GB memory limit and a 30-minute timeout.

Number of Landmarks

Table 5.1 compares the number of causal landmarks found by our method with the state-of-the-art backchaining method used in LAMA (Richter et al., 2008) (RHW). With $m = 1$, our approach is equivalent to the Zhu & Givan procedure, and in accordance with theory generates a superset of the causal landmarks found by the RHW method, increasing the number of landmarks found by 10–30% in several domains. With $m = 2$, the single fluent landmarks found are a superset of those found with $m = 1$, improving on RHW by up to 60%. When conjunctive landmarks found with $m = 2$ are also considered, the number of landmarks found exceeds that of the RHW method by factors between 3 and 43. However, using $m = 2$ is computationally costly. Landmark generation with $m = 2$ times out or runs out of memory in several instances of the Airport and Freecell domains, as well as on many large tasks in other domains that are far beyond the reach of current optimal planners.

	# Causal	Ratio to RHW		
	LMs	**m = 1**	**m = 2**	**m = 2**
Domain	**RHW**	**(Z&G)**	**Single**	**Conj.**
Airport (11)	1043	1.00	1.00	24.07
Blocks (35)	1444	1.00	1.05	8.36
Depot (21)	1379	1.07	1.13	13.11
Driverlog (19)	441	1.02	1.02	6.71
Freecell (54)	4110	1.26	1.27	15.33
Grid (4)	70	1.14	1.14	3.36
Gripper (20)	960	1.00	1.00	10.35
Logistics-1998 (23)	816	1.00	1.00	3.45
Logistics-2000 (28)	1319	1.00	1.00	4.02
Miconic (150)	7720	1.00	1.00	3.52
Mprime (26)	96	1.07	1.67	2.72
Mystery (16)	66	1.03	1.64	2.86
Openstacks (24)	2946	1.03	1.03	11.08
Pathways (30)	954	1.50	1.57	7.32
Pipesw. Not. (44)	754	1.22	1.29	4.25
Pipesw. Tank. (26)	524	1.15	1.24	5.42
PSR Small (50)	550	1.00	1.60	7.32
Rovers (32)	687	1.15	1.17	6.27
Satellite (23)	515	1.01	1.01	7.31
TPP (24)	751	1.13	1.32	5.94
Trucks (14)	467	1.23	1.25	8.92
Zenotravel (18)	309	1.05	1.05	5.25
Elevators (30)	629	1.12	1.12	3.66
Openstacks (30)	2925	1.03	1.03	11.37
PARC Printer (30)	2142	1.00	1.07	18.48
Peg Solitaire (30)	1457	1.00	1.02	19.33
Scanalyzer (26)	673	1.00	1.26	9.65
Sokoban (29)	605	2.73	5.25	43.02
Transport (30)	390	1.00	1.00	3.44
Woodworking (30)	1520	1.06	1.08	9.91

Table 5.1: Number of causal fluent landmarks found by RHW and average ratio to this of the causal landmarks found by our approach. In the second to last column, only single fluent landmarks found with $m = 2$ are counted, the last column includes conjunctive landmarks as well. Top part of table: STRIPS domains of IPC 1–5. Only solvable problems are listed for Mystery. Bottom part of table: domains of the optimal track of IPC 6. Numbers in parentheses show the number of tasks considered for that domain (the tasks where LM generation finished for all configurations).

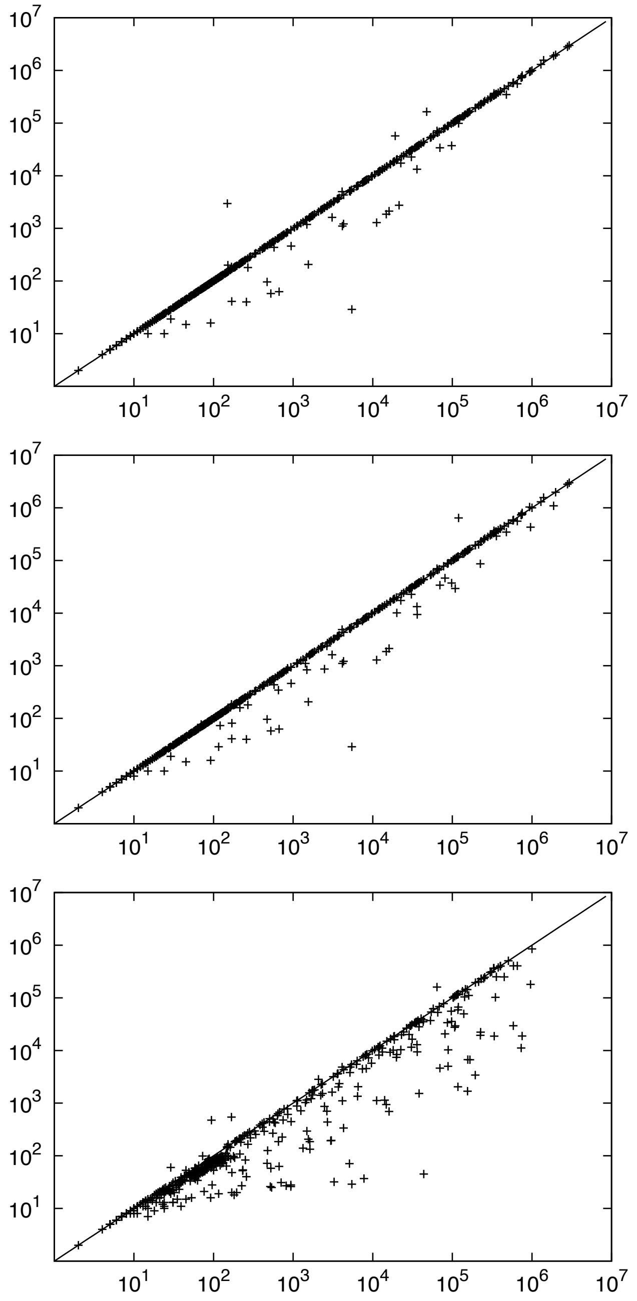


Figure 5.3: Node expansions with optimal cost partitioning, compared to RHW landmark generation (x-axes), of our approach using $m = 1$ (top), $m = 2$ when using only facts (middle), and $m = 2$ when using facts and conjunctive landmarks (bottom). Points below diagonal line are problems in which our approach expands fewer nodes.

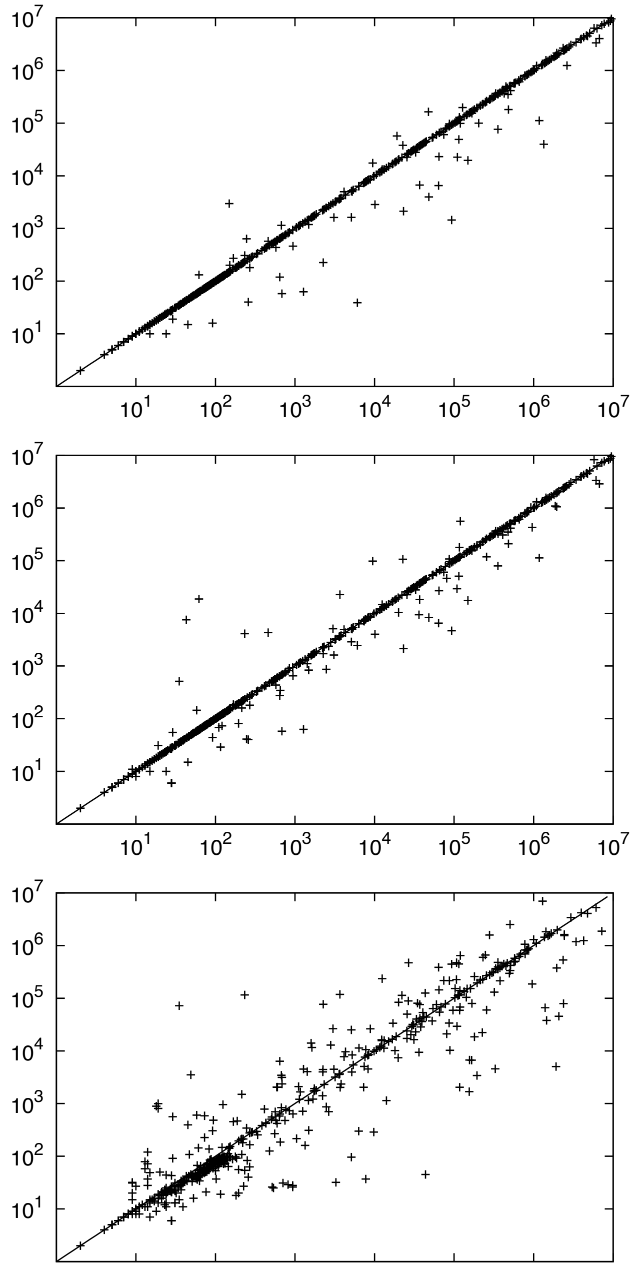


Figure 5.4: Node expansions with uniform cost partitioning, compared to RHW landmark generation (x-axes), of our approach using $m = 1$ (top), $m = 2$ when using only facts (middle), and $m = 2$ when using facts and conjunctive landmarks (bottom). Points below diagonal line are problems in which our approach expands fewer nodes.

	RHW	**Improvement over RHW**		
	# Exp.	**m = 1**	**m = 2**	**m = 2**
Domain		**(Z&G)**	**Single**	**Conj.**
Airport (11)	384	1.00	1.00	1.12
Blocks (23)	2550007	1.00	1.00	7.84
Depot (4)	365373	1.07	1.48	3.75
Driverlog (8)	868496	1.00	1.00	1.02
Freecell (37)	189661	2.14	2.14	2.45
Grid (1)	270	1.50	1.50	1.64
Gripper (5)	458498	1.00	1.00	1.00
Logistics-1998 (3)	45663	1.00	1.00	1.48
Logistics-2000 (20)	862443	1.00	1.00	22.80
Miconic (141)	135213	1.00	1.00	1.34
Mprime (15)	313579	1.00	1.34	1.39
Mystery (12)	290133	1.00	1.00	1.00
Openstacks (7)	27392	1.00	1.00	1.00
Pathways (4)	152448	1.60	1.60	1.60
Pipesw. Not. (16)	1931233	1.05	1.05	1.46
Pipesw. Tank. (8)	29698	1.00	1.00	0.91
PSR Small (48)	697969	1.00	1.03	1.62
Rovers (5)	231520	1.06	1.06	1.06
Satellite (5)	1012920	1.01	1.01	1.08
TPP (5)	12355	1.00	1.00	1.00
Trucks (2)	108132	1.02	1.02	1.05
Zenotravel (8)	186334	1.00	1.00	1.02
Elevators (7)	483982	1.00	1.00	1.35
Openstacks (10)	649341	1.00	1.00	1.00
PARC Printer (12)	1118898	1.00	1.29	1.61
Peg Solitaire (23)	1734655	1.00	1.04	1.20
Scanalyzer (11)	23029	1.00	1.00	1.46
Sokoban (10)	1229907	1.02	1.05	0.90
Transport (9)	929285	1.00	1.00	1.00
Woodworking (10)	199666	1.41	1.41	2.35

Table 5.2: Expanded nodes when using the landmark generation of RHW and average improvement ratios of our approach using the optimal cost partitioning method. In the second to last column, only single fluent landmarks found with $m = 2$ are counted, the last column includes conjunctive landmarks as well. Numbers in parentheses show the number of tasks considered for that domain (the tasks solved by all configurations).

Heuristic Accuracy of Landmark Information

In order to assess how the additional landmarks influence heuristic accuracy, we use them in the LM-A* algorithm with the admissible landmark counting heuristic of Karpas & Domshlak (Karpas and Domshlak, 2009), which we extend to handle conjunctive landmarks. Cost partitioning among landmarks is performed optimally. Table 5.2 shows the number of expanded states in those tasks solved by all configurations. We show results both for when $m = 2$ is used only to compute additional single fluent landmarks, and for when the additional conjunctive landmarks are considered during planning. The number of expansions is improved in some domains by 30–50% even when using only the additional facts found with $m = 2$. With conjunctive landmarks, improvements of factors beyond 2 occur in several domains, with the planner expanding 22 times fewer nodes in the Logistics-2000 domain.

The expansion data from Table 5.2 can be compared with the number of expansions resulting from *uniform* cost partitioning by considering Figures 5.3 and 5.4. While our approach expands significantly fewer nodes than RHW when used in combination with optimal cost partitioning, with uniform partitioning this advantage is smaller for $m = 2$ when using only facts, and all but disappears for $m = 2$ when also using conjunctive landmarks.

Planning Performance

While optimal cost partitioning among landmarks leads to best heuristic accuracy, this method is unfortunately too costly to be competitive with the simpler uniform cost partitioning in terms of runtime and total number of problems solved. In Table 5.3, we report the total number of tasks solved with each of our experimental configurations when using the uniform partitioning method. The number of problems solved in domains in which landmark generation with $m = 2$ was computationally too costly (timing out in tasks that were solved by RHW) are shown in parentheses at the bottom of the table and not included in the total. Our approach with $m = 1$ solves slightly more tasks than RHW, and $m = 2$ using only single fluent landmarks solves one more task than $m = 1$. Using conjunctive landmarks during planning, however, does not improve coverage.

The results in this table are not as good as could be expected when considering the improvement in expanded states shown in Table 5.2. The scatter plots in Figures 5.3 and 5.4 indicate that this may be due in large part to the uniform cost partitioning method.

Table 5.4 shows detailed results for selected domains, demonstrating how the benefit of additional heuristic accuracy does not always pay off compared to the extra computational effort needed for generating and managing the conjunctive landmarks. While in Logistics-2000, our approach using $m = 2$ performs better than RHW both with respect to expansions and time, in Depot, $m = 2$ performs better with respect to expansions, but worse with respect to time. Driverlog is an example where the conjunctive landmarks are not helpful at all and RHW performs better both with respect to expansions and time. We also found that while having more causal fact landmarks *usually* translates to better heuristic accuracy, this is not always the case when using the uniform cost partitioning scheme.

5.5 Related Work

Landmarks were first introduced in the context of the relaxed planning graph, in which they are computed via the backchaining procedure (Porteous et al., 2001; Hoffmann et al., 2004). As the backchaining procedure alone does not find a large number of landmarks, it is enhanced with a lookahead step in which intermediate disjunctive goals are backchained from and then discarded. One recent innovation in the computation of landmarks with backchaining is to generalize landmarks to formulas over fluents and adapt landmark utilization algorithms to directly consider disjunctions over the set of fluents (Richter et al., 2008). The computation of landmarks by forward propagation of sets in the relaxed planning graph was first introduced by Zhu and Givan (2003). This allowed the complete set of landmarks for the delete relaxation to be found, and is one method

Domain	RHW	m = 1 (Z&G)	m = 2 Single	m = 2 Conj.
Blocks (35)	26	26	26	28
Depot (22)	7	7	7	7
Driverlog (20)	10	10	10	9
Grid (5)	2	2	2	2
Gripper (20)	7	7	7	7
Logistics-1998 (35)	3	3	3	3
Logistics-2000 (28)	20	20	20	22
Miconic (150)	142	142	142	142
Mystery (19)	15	15	15	15
Openstacks (30)	7	7	7	7
Pathways (30)	4	4	4	4
Pipesw. Not. (50)	19	19	19	18
Pipesw. Tank. (50)	12	13	13	11
PSR Small (50)	49	49	49	49
Rovers (40)	6	6	6	5
Satellite (36)	6	6	6	6
TPP (30)	6	6	6	6
Trucks (30)	2	2	2	2
Zenotravel (20)	8	8	8	8
Elevators (30)	13	13	13	14
Openstacks (30)	17	17	17	12
PARC Printer (30)	14	14	16	12
Peg Solitaire (30)	27	27	27	25
Scanalyzer (30)	9	9	9	6
Sokoban (30)	21	24	23	14
Transport (30)	11	11	11	11
Woodworking (30)	13	12	12	9
Total (951)	476	479	480	454
(Airport) (50)	(26)	(26)	(11)	(11)
(Freecell) (80)	(55)	(60)	(49)	(30)
(Mprime) (35)	(19)	(19)	(19)	(19)

Table 5.3: Solved problems when using the landmark generation of RHW and our approach using the uniform cost partitioning method. In the second to last column, only single fluent landmarks found with $m = 2$ are counted, the last column includes conjunctive landmarks as well. Numbers in parenthesis show the total number of solvable tasks in that domain.

	RHW			m = 2 using conj. LMs		
Inst.	**LM**	**Exp.**	**Time**	**LM**	**Exp.**	**Time**
Logistics-2000						
5-0	33	936	0.06	33 + 66	28	0.01
7-0	44	7751	0.58	44 + 112	37	0.03
10-0	56	194038	21.85	56 + 192	3421	3.69
11-0	61	156585	24.00	61 + 221	6706	7.65
12-0	56	117387	16.91	56 + 236	2041	2.82
Depot						
2	29	1488	0.08	34 + 193	310	0.18
4	54	2347873	220.95	60 + 111	531785	1237.57
7	42	167561	13.35	46 + 351	79755	78.00
10	47	1956533	197.43	55 + 82	375300	578.99
13	62	507369	77.64	62 + 625	336331	822.89
Driverlog						
3	10	1109	0.04	10 + 29	2105	0.10
5	17	247579	9.65	17 + 73	658799	73.10
7	17	26591	1.54	17 + 94	88915	19.66
10	14	504955	24.29	14 + 55	2506690	324.94
11	14	1298547	49.62	14 + 49	6969276	690.56

Table 5.4: Detailed results for select domains, comparing $m = 2$ to RHW with respect to landmarks found, expanded states and runtime. Landmarks shown are causal facts for both approaches and conjunctive landmarks for $m = 2$ (second term in the sum).

by which the solution to the equations we introduce above can be computed.

Landmarks were initially used as search control information in the form of a control loop, in which a planner is repeatedly invoked with the next landmark to be achieved until the global goal of the problem is reached (Hoffmann et al., 2004). While this method results in increased coverage, the quality of the plans generated suffers as the planner ignores other subgoals while generating a plan for the landmark currently being considered. Additionally, a planner constructed in this way may be incomplete even if the planner invoked to find solutions to each subproblem is guaranteed to eventually find a solution if one exists. A second approach to landmark utilization that maintains completeness is to compute a heuristic estimate using landmarks, and use this estimate in the context of a complete search algorithm. One such heuristic propagates in the relaxed planning graph *counts* of the number of times a landmark must be achieved that take into account the delete effects of operators (Zhu and Givan, 2003), then solving a generalized bin-packing problem in which every separate achievement of a landmark must be associated with an action. The number of actions required for all achievements of the remaining landmarks then constitutes the heuristic value for a state. A more recent heuristic based on counting landmarks uses the idea of greedy-necessary orderings between delete-relaxation landmarks to compute landmarks that must be achieved multiple times, and has been shown to significantly increase the number of problems solved with heuristic search when used in combination with another heuristic (Richter et al., 2008). This heuristic makes the inference that a landmark which has previously been achieved, yet is not currently true, must be reachieved if it is ordered greedy-necessarily before a landmark that has not yet been achieved.

More recently, much work has been directed at the use of landmarks in optimal planning, both through the formulation of admissible heuristics and modifications to existing optimal search algorithms. One way in which an admissible heuristic can be obtained from landmarks is to use a *cost-partitioning function* to distribute the cost of each action in the problem among all of the landmarks which it can achieve (Karpas and Domshlak, 2009). It can be shown that the heuristic estimator resulting from summing the minimum cost of achieving each landmark over the full set of known landmarks then results in an admissible heuristic. One important direction of future research in this area is the formulation of cost-partitioning functions that are less computationally costly to compute, yet nevertheless result in informative heuristic values. A second approach to formulating admissible heuristics is the use of delete-relaxation landmarks in order to closely approximate the optimal cost of the delete relaxation (Helmert and Domshlak, 2009). This idea is based on finding cuts induced by landmarks in the delete relaxation graph, and summing the cost of these cuts with the cost of the rest of the graph, computed by recursively applying the same procedure. This admissible estimate of the cost of the delete relaxation can be further improved by considering several cuts simultaneously and treating the sets of actions as the components of a hitting set problem (Bonet and Helmert, 2010).

Additionally, in the presence of path-dependent heuristics such as those resulting from landmark counting, the A* algorithm can be modified to take into account the fact that paths along which certain landmarks have not been achieved may assign higher (admissible) costs to a state than other paths to the same state. As both estimates are guaranteed to be admissible, the higher of these two heuristic values can then be used as the estimate for the state. The algorithm resulting from these improvements is called LM-A*.

5.6 Conclusions

We have shown how to declaratively define the complete set of causal landmarks for the delete relaxation. Given this definition, the complete set of causal delete relaxation landmarks can be computed in polynomial time using propagation algorithms similar to those used to compute the heuristics discussed in previous chapters. Combined with the Π^m compilation that yields a problem with no deletes which nevertheless encodes information about deletes in its fluents and operators, we obtain a parameterised method that permits the computation of conjunctive and single fact landmarks that take into account delete information in planning problems. The experimental evaluation of these landmarks indicates that their use can significantly increase the informativeness of landmark-based admissible heuristics.

Chapter 6

Heuristics with Choice Variables

In this chapter we introduce a heuristic based on the idea of choice variables, implicit multivalued variables to which every valid plan must assign at most one value. We show that by reasoning by cases over different assignments to such variables, heuristic estimates can be improved in certain problems, and adapt an algorithm from the field of graphical models to our setting to perform such reasoning efficiently. The resulting formulation shares some aspects with a family of approaches known as factored planning, offering a new interpretation of these techniques.

6.1 Introduction

Many recently proposed techniques in planning make use of *invariants*, which can potentially take advantage of problem structure to a greater extent than delete-relaxation based heuristics. These invariants come in many forms: Landmarks, for example, are invariants over the state trajectories followed by all valid plans that achieve a goal (Chapter 5), and have been used both for search control (Hoffmann et al., 2004; Karpas and Domshlak, 2009) and in the computation of heuristic estimates (Richter and Westphal, 2010; Helmert and Domshlak, 2009). Mutexes in STRIPS planning are sets of fluents that can be shown not to be achievable in conjunction by any plan (Haslum and Geffner, 2000), and have been used to reason about causal constraints in planning (Vidal and Geffner, 2006; Lipovetzky and Geffner, 2009) as well as to extract multivalued representations from STRIPS problems and use them in heuristics that go beyond the delete relaxation in computing their estimates (Helmert, 2006). Here, we consider a new type of invariant in planning: implicit multivalued variables to which a value can be assigned *at most* once by every plan. We refer to such variables as *choice variables.* Choice variables are dissimilar to standard multi-valued variables in that their values do not represent a property of the current state that may change in future states, but rather a commitment made by the planner to solve the problem while accepting the constraints that a choice imposes upon the space of possible plans. Indeed, they are more similar in spirit to the variables that are used in graphical models such as constraint satisfaction problems (CSPs) and Bayesian networks. In these types of problems, a single value that is consistent with the rest of the solution to the problem, or a single most likely instantiation, must be chosen for each variable. Such variables can be represented in STRIPS as sets of fluents, each corresponding to one value of the implicit multivalued choice variable, of which at most one can be made true in any valid plan.

While such variables in CSPs typically impose hard restrictions on each other's values, in the planning setting the more frequent case is that making an assignment to one of them influences the *cost* of making an assignment to another, or of solving the overall problem. Choice variables can then be used as a tool for improving the quality of planning heuristics, by forcing heuristics to respect features of a problem that otherwise might not be present in problem relaxations. For a simple example, consider a scenario in which an agent must purchase various items at a market. There are various markets from which all of the items can be purchased at different prices, but the agent can only make one trip and must choose a single market at which to purchase all of them. A multivalued variable whose domain consists of the set of markets and represents the choice made by the agent for where to buy the items then constitutes a choice variable to which a value must be assigned by every valid plan, and which once assigned, can not be changed. A STRIPS encoding of such a problem might contain operators $o = \langle \mathit{Pre}(o), \mathit{Eff}(o) \rangle$ such as the following:

- $\mathit{goto\text{-}market}_i = \langle \{\mathit{at\text{-}home}\}, \{\neg \mathit{at\text{-}home}, \mathit{at\text{-}market}_i\} \rangle$
- $\mathit{buy\text{-}item}_{ij} = \langle \{\mathit{at\text{-}market}_i\}, \{\mathit{have\text{-}item}_j\} \rangle$

with the cost of each of the $\mathit{buy\text{-}item}_{ij}$ operators being the cost of item i at market j. In the delete relaxation Π^+ of this problem, the delete-relaxed operators $\mathit{goto\text{-}market}_i^+ = \langle \{\mathit{at\text{-}home}\}, \{\mathit{at\text{-}market}_i\} \rangle$ do not delete the fluent *at-home*, and the fact that once one market is chosen the agent can no longer go to other markets is no longer encoded. The optimal delete relaxation plan will therefore be to apply the $\mathit{goto\text{-}market}_i^+$ operator for all markets i that carry some required item at the minimum price, and buy the items at different markets, leading to a large underestimation of the real cost of the problem. However, the knowledge that the set of markets constitutes a choice variable allows us to improve the informativeness of the heuristic by reasoning independently about each of its possible values. This can be done by computing the value of a delete relaxation heuristic in several different versions of the same problem, in each of which the operators that allow moving to all but one of the markets are disallowed. Estimating the cost of each of these subproblems and taking the *minimum* among them then allows us to obtain an improved estimate for the problem as a whole.

While the technique of enumerating all possible assignments to the set of choice variables may be feasible in problems with few such variables, the number of complete assignments that must be considered grows exponentially in their number. For example, consider a variant of the problem above in which several markets from which to buy various *classes* of goods must be chosen in a specified order, and the action of moving between each of the markets is associated with a cost. The agent might first have to buy groceries from one of a set of markets that sells food, then move to one of a set of markets that sells hardware in order to buy light bulbs, and so on. The possible assignments that need to be considered is then the product of the number of markets of each type. However, the choice of at which market to buy each type of good is *independent* of all of the other markets except for that of the type of good that directly precedes it, and explicitly enumerating each global assignment is not necessary to obtain the optimal cost. Such a problem constitutes a *cost network* in which the interaction graph representing how the assignment made to each choice variable affects the cost of making some assignment to the others is a *chain*. To take advantage of this kind of problem structure, we turn to the field of graphical models, in which the problem of exponential growth in the number of possible global assignments is a familiar one, and focus on a family of methods known as *conditioning* methods. Conditioning methods exploit the fact that the interaction between two variables may be reduced, or the two variables made independent, by instantiating some subset of the variables that lie along the paths between them in the interaction graph of the problem (Pearl, 1988; Boutilier et al., 1996). The problem can then be solved by enumerating the assignments to this subset and solving the remaining parts of the problem, now rendered independent of one another or their interactions made easier to reason about, separately. Here we adapt the optimal *recursive conditioning* algorithm (Darwiche, 2001) to perform this type of reasoning in the planning setting, computing the minimum heuristic cost of a state over all possible assignments to a set of choice variables.

The resulting algorithm can also be seen as an instance of the *factored* approach to planning. Factored planning methods decompose a planning problem into a set of *factors* or *components*, and attempt to find plans for each that can be interleaved with the plans for the others in order to obtain a global solution (Amir and Engelhardt, 2003; Brafman and Domshlak, 2006; Fabre et al., 2010). Such subproblems may share with each other either fluents or actions; our approach is similar to the former, with the fluents representing different values of the choice variables of the problem being shared between components. Each of the subproblems that we consider then consists of making an assignment to a *target* choice variable C_t, given an assignment to the set of choice variables on which C_t is directly dependent. However, while factored planning methods attempt to find explicit plans for the global problem, we instead obtain heuristic estimates of the cost of solving each subproblem and combine these costs to obtain a heuristic for the global problem. If actions are also shared between components, for instance, the result is a less informative heuristic estimate, and not an unsound planning algorithm.

In the remainder of this chapter, we concentrate on how to use choice variables to efficiently compute better heuristic estimates for planning problems, and assume that the choice variables themselves are given to the planner along with the domain and problem representations in the form of a set of subsets of the fluents of a problem, with each subset representing the possible values of a choice variable. Techniques for automatically identifying sets of fluents that naturally have the choice variable property are a topic of further research.

6.2 The Choice Variables Heuristic

We define choice variables formally as follows:

Definition 6.1 (Choice variable)**.** *A* choice variable $C = \{\perp, d_1, \ldots, d_n\}$ *in a planning problem is a multi-valued variable whose value* $\perp$ *denotes that it is unassigned, and which can take on at most one of the values in* $\{d_1, \ldots, d_n\}$ *during the execution of any plan.*

Definition 6.2 (Choice variable in STRIPS)**.** *A choice variable* C *in* STRIPS *consists of a set of fluents* $\{d_1, \ldots, d_n\}$*. When none of these fluents are true, the value of the implicit multivalued variable is* $\perp$*. No operator in the problem may add more than one of these fluents, and those operators adding exactly one must have effects that make the preconditions of other such operators unachievable.*

Given a planning problem Π in STRIPS with a set of choice variables $C = \{C_1, \ldots, C_n\}$, each with domain $D(C_i)$, we denote with $\phi(C)$ the set of possible assignments to C, where an assignment is a function $\mathbf{v}$ that maps each variable C_i for which it is defined to a value in its domain $D(C_i)$. We use the notation $\mathbf{v}[C_i]$ to denote the value assigned to C_i by $\mathbf{v}$. For an assignment $\mathbf{v}$, a problem $\Pi^{\mathbf{v}}$ that *respects* this assignment can be obtained by removing from the problem all operators that assign a value to any choice variable C_i that is inconsistent with $\mathbf{v}$:

Definition 6.3 ($\Pi^{\mathbf{v}}$)**.** *Given a problem $\Pi = \langle F, I, O, G\rangle$, a set of choice variables C, and an assignment $\boldsymbol{v}$ to C, let $\Pi^{\boldsymbol{v}} = \langle F, I, O^{\boldsymbol{v}}, G\rangle$, where $O^{\boldsymbol{v}}$ is the set of operators in O that* respect *$\boldsymbol{v}$ and is given by*

$$O^{\boldsymbol{v}} = \bigcap_{C_i \in C} \{o \in O \mid (D(C_i) \setminus \{\boldsymbol{v}(C_i)\}) \cap \mathit{Add}(o) = \emptyset\} \tag{6.1}$$

In other words, $O^{\mathbf{v}}$ consists of the set of operators that for each choice variable C_i either make no assignment to it or add the value in $D(C_i)$ that is specified by $\mathbf{v}$. Note that in the $\Pi^{\mathbf{v}}$ problem the values of each choice variable $\mathbf{v}(C_i)$ implied by the assignment $\mathbf{v}$ are *not* added to the initial state I. The optimal plan for $\Pi^{\mathbf{v}}$ is then the lowest cost plan for Π that chooses the values for each $C_i \in C$ implied by the assignment $\mathbf{v}$.

Given a *base heuristic* h, we can define the *choice variables heuristic* h^c in terms of this heuristic and the restricted problems $\Pi^{\mathbf{v}}$ for $\mathbf{v} \in \phi(C)$:

$$h^c = \min_{\mathbf{v} \in \phi(C)} h(\Pi^{\mathbf{v}}) \tag{6.2}$$

Proposition 6.4 (Admissibility and optimality of h^c)**.** *Given a problem Π with a set of choice variables C and an admissible base heuristic h, h^c is also admissible. Furthermore, if h is the perfect heuristic h^*, then $h^c = h^*$.*

Proof. Let π^* be an optimal plan for Π, and $\mathbf{v}$ the assignment to C made by π^*. Then π^* also constitutes a plan for $\Pi^{\mathbf{v}}$, since it cannot contain any operators assigning values to C other than those in $\mathbf{v}$. If h is admissible, we have that $h^c(\Pi) \leq h(\Pi^{\mathbf{v}}) \leq \mathit{cost}(\pi^*)$, and $h^c(\Pi)$ is admissible. If $h = h^*$, $h(\Pi^{\mathbf{v}}) = \mathit{cost}(\pi^*)$, and there can be no $\mathbf{v}'$ for which $h(\Pi^{\mathbf{v}'}) < cost(\pi^*)$, as any plan for $\Pi^{\mathbf{v}'}$ is a plan for Π as well and this would contradict the optimality of π^*. Then $h^c(\Pi) = h(\Pi^{\mathbf{v}}) = cost(\pi^*)$, and $h^c(\Pi) = h^*(\Pi)$. □

While the value of h^c can be calculated as implied by Equation 6.2, this results in the number of evaluations of the base heuristic being exponential in the number of choice variables $|C|$, or more precisely, equal to the number of possible different instantiations of the set C, $|\phi(C)|$. For problems in which choice variables exhibit some degree of conditional independence, however, it may be possible to reduce the number of calls to h by taking advantage of problem structure. We now describe how to obtain a graph with choice variables as nodes which encodes the causal dependencies between them, and show how to adapt to our setting the *recursive conditioning* algorithm, first introduced in the context of Bayesian networks, to take advantage of these relationships.

The Choice Variables Graph

Given a problem Π with choice variable set C, we encode its structure by constructing a directed graph $G_C = \langle C \cup C_G, E_C \rangle$, where $C_G = \cup_{g \in G}\{C_g\}$ is the set of *goal choice variables*, each of whose domains are $D(C_{g_i}) = \{g_i\}$, where $G = \{g_1, \ldots, g_m\}$ is the set of goals of the problem. In the following, C_i may denote either one of the explicitly defined choice variables of the problem or one of these goal choice variables. An edge $e = \langle C_i, C_j \rangle$ in G_C signifies that the cost of making some assignment to the choice variable C_j is directly dependent upon the assignment made to C_i, while the lack of such an edge signifies that the cost of setting C_j to any one of its values is *conditionally independent* of the assignment made to C_i given the assignments made to its set of parent nodes $Par(C_j) = \{C_k \mid \langle C_k, C_j \rangle \in E_C\}$. To describe the construction of G_C, we first define the notion of *relevance* between the fluents of a planning problem:

Definition 6.5 (Relevance). *Given a planning problem* $\Pi = \langle F, I, O, G \rangle$, $p \in F$ *is* relevant *to* $q \in F$ *if:*

- *$p = q$, or*
- *There exists $o \in O$ such that $p \in Pre(o)$ and $r \in Add(o) \cup Del(o)$, and r is relevant to q.*

Intuitively, the relevance relation tells us that the cost of making a fluent q true or false in a future state is dependent on whether p is true in the current state or not. *Conditional relevance* given a set of fluents B can be defined similarly:

Definition 6.6 (Conditional relevance)**.** *$p \in F$ is* conditionally relevant *to $q \in F$ given $B \subseteq F$ if:*

- *$p = q$, or*
- *There exists $o \in O$, $r \in (F \setminus B)$ such that $p \in Pre(o)$ and $r \in Add(o) \cup Del(o)$, and r is conditionally relevant to q given B.*

In this case, the intuition is that p is conditionally relevant to q given B if p somehow influences the cost of achieving q, even when it is known whether all fluents in B are currently true or false. These definitions can easily be extended to sets of fluents, with a set $P \subseteq F$ being (conditionally) relevant to $Q \subseteq F$ (given B) if any $p \in P$ is (conditionally) relevant to any $q \in Q$ (given B). Denoting this with $rel(P, Q, B)$, the set of edges E_C of G_C is given by:

$$E_C = \{\langle C_i, C_j \rangle \mid rel(D(C_i), D(C_j), \cup_{C_k \in C \setminus \{C_i, C_j\}} D(C_k))\}$$

In words, there is an edge $e = \langle C_i, C_j \rangle$ in G_C if $D(C_i)$ is conditionally relevant to $D(C_j)$ given the values of all other choice variables. In what follows, we will assume that the choice variables graph G_C given by this definition is *acyclic.* We will later investigate the implications of relaxing this assumption.

G_C encodes a set of relationships between its nodes similar to those encoded by a Bayesian network, but rather than associating with each node C_i a *conditional probability table* (CPT) which specifies in each entry the probability of each node's taking some value in its domain given the values of its parent nodes, it associates with each node a *conditional cost table* (CCT) that specifies in each entry the cost of the problem of assigning to C_i some value $d \in D(C_i)$ given an assignment to its parent nodes $Par(C_i)$. As a STRIPS problem, this is the cost of the goal $G = \{d\}$ from an augmented initial state $s \cup \{\mathbf{v}^{Par(C_i)}\}$, where $\{\mathbf{v}^{Par(C_i)}\}$ denotes the values of the assignment $\mathbf{v}$ for the set of choice variables $Par(C_i)$. The operators of this problem are restricted to the set $O^{\mathbf{v}}$, which is computed as in Definition 6.3. Differently from Definition 6.3, which specifies a problem whose cost is the cost of the whole problem given certain assumptions, here we wish to isolate the cost of each assignment to a choice variable in a single subproblem, and therefore include the fluents representing the assignment made to the set $Par(C_i)$ in the initial state of the problem.

In the CCT, the costs of each of these problems can be represented implicitly by means of an *assignment-respecting heuristic* that given an assignment $\mathbf{v}$ is able to estimate the cost of the modified planning problem in which only those operators that *respect* $\mathbf{v}$ can be used. Existing delete-relaxation heuristics such as those presented in Chapters 3 and 4 can easily accommodate this requirement by simply not considering those operators when propagating costs or relaxed plans. Without making any assumptions about the specific heuristic used, we denote the values of a heuristic respecting an assignment $\mathbf{v}$ by $h^{\mathbf{v}}$, and write the estimated cost of such a problem $h^{\mathbf{v}}(C_i = d \mid Par(C_i))$. This notation parallels that used in Bayesian nets for the conditional probability of a node being instantiated to a value d given the values of its parent nodes, and makes clear the relationship between the subproblems considered here and families in Bayesian nets that consist of a node together with all of its parents. The *choice variables decomposition heuristic* h^{cd} can then be written as follows:

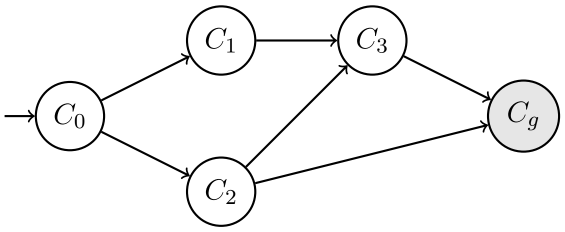


Figure 6.1: A choice variable graph.

$$h^{cd}(\Pi) = min_{\mathbf{v}\in\phi(C)} \left[\sum_{i=1}^{|G|} h^{\mathbf{v}}(C_{g_i} = g_i \mid Par(C_g)) \;+ \right. \tag{6.3}$$
$$\left. \sum_{i=1}^{|C|} h^{\mathbf{v}}(C_i = \mathbf{v}[C_i] \mid Par(C_i)) \right]$$

Note that while all possible assignments to non-goal choice variables are considered, the values of the goal choice variables are forced to take on their (unique) values g_i. This can be seen as analogous to the notion of *evidence* in Bayesian nets, which are nodes whose values have been observed and which can therefore not take on other values.

As an example, consider the choice variable graph of a problem with a single goal g shown in Figure 6.1. h^{cd} would estimate the cost of this problem in terms of the 5 subproblems corresponding to setting the values of C_0, C_1, C_2, C_3, and C_g. The value of the heuristic would then be given by:

$$h^{cd}(\Pi) = min_{\mathbf{v}\in\phi(C)} \Big[h^{\mathbf{v}}(C_g = g \mid C_2, C_3) + h^{\mathbf{v}}(C_0 = \mathbf{v}[C_0]) +$$
$$h^{\mathbf{v}}(C_1 = \mathbf{v}[C_1] \mid C_0) + h^{\mathbf{v}}(C_2 = \mathbf{v}[C_2] \mid C_0) +$$
$$h^{\mathbf{v}}(C_3 = \mathbf{v}[C_3] \mid C_1, C_2) \Big]$$

In addition to the acyclicity of G_C, Equation 6.3 makes one more assumption that must be true for the heuristic to uphold the admissibility and optimality guarantees of Proposition 6.4. This is that the cost of no operator is counted in more than one subproblem. To formalize this criteria, we extend the definitions of relevance and conditional relevance to operators: we say that operator o is (conditionally) relevant to fluent p (given A) if it adds or deletes some fluent that is (conditionally) relevant to p (given A). For the decomposition of Equation 6.3 to preserve admissibility and optimality, it is then necessary that there is no more than one C_i for each operator o with $cost(o) > 0$ such that o is conditionally relevant to C_i given the set $\bigcup_{C_j \in C \setminus \{C_i\}} D(C_j)$. When this condition is not met, the cost of o may be counted in more than one subproblem, leading to an inadmissible heuristic even if the underlying heuristic used to compute the costs of subproblems is admissible. There is a parallel to be drawn here between the behaviour of h^{cd} and that of the additive heuristic h^{add}: in both cases, the cost of an action may be overcounted when it is used in the (relaxed) plan in order to achieve more than one subgoal whose cost is counted towards the cost of the overall problem.

Proposition 6.7 (Admissibility and optimality of h^{cd})**.** *Given a problem* Π *with a set of choice variables C such that the choice variable graph G_C is directed acyclic and each operator o of the problem is conditionally relevant to at most one choice variable given the other choice variables, and an admissible base heuristic h, the decomposition h^{cd} of Equation 6.3 is also admissible. Furthermore, if $h = h^*$, then $h^{cd} = h^*$.*

Proof. Let π^* be an optimal plan for Π, $\mathbf{v}$ the assignment to C made by π^*, and π_i^* a minimal subsequence in π^* that constitutes a plan for the subproblem of setting $C_i = \mathbf{v}[C_i]$. Such a subsequence must exist, since every valid plan assigns a value to all $C_i \in C$ by definition. Furthermore, the sets of operators with non-zero cost in all π_i^* must be disjoint, since each such operator is conditionally relevant to at most one choice variable, and an operator that is not conditionally relevant to C_i can be removed from π_i^* to give a smaller plan for the subproblem. Finally, each π_i^* must be optimal, since otherwise it could be replaced with a lower cost π_i resulting in a global plan of lower cost. If h is admissible, then

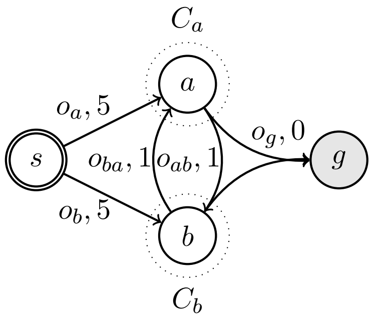
$$\begin{aligned}
h^{cd}(\Pi) &= \sum_{i=1}^{|G|} h^{\mathbf{v}}(C_{g_i} = g_i \mid \mathit{Par}(C_{g_i})) + \\
&\quad \sum_{i=1}^{|C|} h^{\mathbf{v}}(C_i = \mathbf{v}[C_i] \mid \mathit{Par}(C_i)) \\
&\leq \sum_{i=1}^{|G|} \mathit{cost}(\pi^{*\mathbf{v}}(C_{g_i} = g_i \mid \mathit{Par}(C_{g_i}))) + \\
&\quad \sum_{i=1}^{|C|} \mathit{cost}(\pi_i^{*\mathbf{v}}(C_i = \mathbf{v}[C_i] \mid \mathit{Par}(C_i))) \\
&= \mathit{cost}(\pi^*)
\end{aligned}$$

and $h^{cd}(\Pi)$ is also admissible. If $h = h^*$, a similar argument applies and $h^{cd}(\Pi)$ is optimal. □

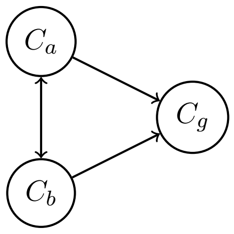
We now consider the values of h^{cd} when cycles are present in G_C. The equation then encodes that each choice variable C_i in the cycle can rely on some assignment to its set of parent variables $\mathit{Par}(C_i)$, and that in turn one of those variables may directly or indirectly rely on an assignment to C_i, violating the well-foundedness of the implicit global plan. To see how this can affect the value of the heuristic, consider Figure 6.2. Delete effects that ensure that a and b can only be set once have been omitted for simplicity. Since each choice variable $C_a = \{a\}$, $C_b = \{b\}$ has only a single value in its domain, there is a single possible instantiation $\mathbf{v}$ of C. The optimal cost for this problem is 6, resulting from instantiating either C_a or C_b with the cost 5 operator that has only s in its precondition, followed by applying either o_{ab} or o_{ba} with cost 1. However, using h^* as our underlying heuristic in the definition of h^{cd} we obtain:

$$\begin{aligned}
h^{cd}(s) &= h(C_g = g \mid C_a, C_b) + h(C_a = a \mid C_b) + h(C_b = b \mid C_a) \\
&= 0 + 1 + 1 = 2
\end{aligned}$$

in which the heuristic computation essentially encodes that the instantiation of C_a relies on the instantiation of C_b and vice



(a) Problem with choice variables $C_1 = \{a\}$, $C_2 = \{b\}$.



(b) Resulting choice variables graph G_C with two node cycle consisting of C_a, C_b.

Figure 6.2: A planning problem with a cyclic choice variable graph.

versa. By extension of this example it can be seen that while the acyclicity of G_C is not a requirement for the admissibility of h^{cd}, it can result in estimates of lower cost than the optimal plan, and is therefore a requirement for its optimality.

Computing h^{cd} Efficiently

We now turn our attention to how to take advantage of the conditional independence relations between choice variables implied by Equation 6.3, showing how to adapt the recursive conditioning algorithm, originally defined as a method for exact inference on Bayesian networks (Darwiche, 2001), to the planning setting. Given a Bayesian network $\mathcal{N}$ and evidence $\mathbf{e}$ in the form of an observed instantiation of a subset of the nodes of $\mathcal{N}$, the recursive conditioning algorithm operates by selecting a *cutset* C that when instantiated results in two connected components $\mathcal{N}^l, \mathcal{N}^r$ that are *conditionally independent* of one another given C. The method then enumerates the possible instantiations $\mathbf{c}$ of C, recording each and solving $\mathcal{N}^l$ and $\mathcal{N}^r$ by applying the same method recursively with the pertinent evidence. In enumerating the possible instantiations of the cutset, the values for each node given in the evidence $\mathbf{e}$ are respected. When the net-

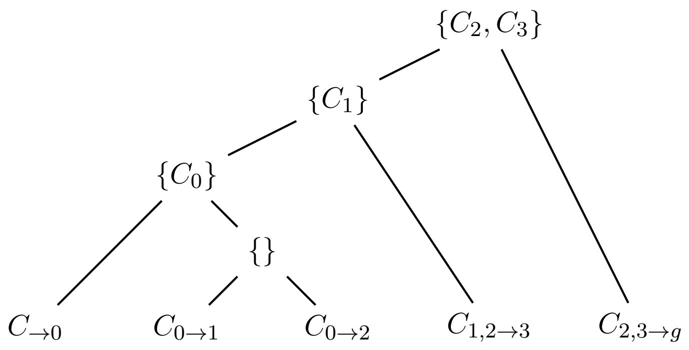


Figure 6.3: A dtree for the choice variables graph shown in Figure 6.1 resulting from the elimination ordering $\langle C_0, C_1, C_2, C_3, C_g\rangle$. Internal nodes are marked with cutsets.

work on which the method is called consists of a single node, the probability corresponding to the current instantiation of the node's parents can be looked up directly in the associated CPT. Given two conditionally independent networks $\mathcal{N}^l, \mathcal{N}^r$, the joint probability given an instantiation $\mathbf{c}$ of the cutset is calculated by multiplying together the probabilities calculated for each, and these results are summed over the set of all possible instantiations of C.

The recursive conditioning algorithm can be driven by a structure known as a *dtree*, a binary tree whose leaves represent the *families* in $\mathcal{N}$, where a family is a node together with its set of parent nodes (Darwiche, 2001). The resulting algorithm is shown in Figure 6.4. Each internal node d of the dtree corresponds to a subnetwork $\mathcal{N}^d$ of $\mathcal{N}$, with the root node corresponding to the full network, and specifies a cutset consisting of those variables shared between its two children d^l and d^r which do not appear in its *acutset*, defined as the set of variables that appear in the cutsets of the ancestors of d. A dtree for the choice variable graph of Figure 6.1 is shown in Figure 6.3. An instantiation of all of the variables that appear in a node's cutset and acutset then ensures that all variables shared between its two children are instantiated, resulting in two networks which are conditionally independent given the instantiation. A dtree node specifies that recursive conditioning be applied to the associated network by instantiating its cutset, and then applying recursive conditioning to the resulting two networks $\mathcal{N}^l$ and $\mathcal{N}^r$ by using d^l and d^r as dtrees for those.

```
Input: A dtree node $D$
Input: An assignment $\mathbf{e}$
Output: The probability of the evidence $\mathbf{e}$

function RC ($D$, $\mathbf{e}$) begin
    if $D$ is a leaf node then
        $X \leftarrow$ variable associated with $D$
        if $X \in \boldsymbol{e}$ then
            return $Prob(\mathbf{e}[\mathbf{X}] \mid \mathbf{e}^{Par(X)})$
        else
            return 1
    else if $cache_D[\mathbf{e}^{context(D)}] \neq undefined$ then
        return $cache_D[\mathbf{e}^{context(D)}]$
    else
        $p \leftarrow 0$
        for $\boldsymbol{c} \in \phi(cutset(D))$ do
            $p \leftarrow p + \mathtt{RC}(D^l, \mathbf{e} \cup \mathbf{c}) \cdot \mathtt{RC}(D^r, \mathbf{e} \cup \mathbf{c})$
        $cache_D[\mathbf{e}^{context(D)}] \leftarrow p$
        return $p$
end
```

Figure 6.4: Recursive conditioning for calculating probability of evidence **e** in a Bayesian network.

To avoid repeated computations, each dtree node may also maintain a *cache*, which records for each instantiation **y** of its *context*, defined as the intersection of the set of variables of the corresponding subnetwork with the nodes of its acutset, the probability resulting from **y**. While this increases the space requirements of the algorithm, in many settings the associated gains in time are of greater importance. Given an *elimination ordering* of width w for a network with n nodes, a dtree with width $\leq w$ can be constructed in linear time.[1] With full caching, recursive conditioning driven with such a dtree is then guaranteed

[1]Elimination orderings and width are beyond the scope of this thesis. For an overview of the subject, see Bodlaender (1993).

to run in $O(n^w)$ time. While finding the optimal elimination ordering for a graph which results in the lowest width dtree is an NP-hard problem, many greedy heuristics provide reasonable performance in practice (Fattah and Dechter, 1996).

In order to calculate the value of h^{cd} using recursive conditioning, we replace the CPTs associated with each family of the graph with CCTs represented implicitly by a *heuristic estimator* h, which rather than calculating the probability of some instantiation of the node given the instantiation of its parent nodes, estimates the *cost* of setting a choice variable to a certain value given the instantiations of its parents in the choice variable graph G_C. To obtain the cost of the planning problem resulting from instantiating the cutset of the choice variable graph associated with the current node, the costs of the two components are *summed* rather than multiplied, and the *minimum* such cost is taken over all of the possible instantiations of the cutset. The facts that the choice variables C_{g_i} have only a single value g_i and hence must be assigned this value, and that choice variables may already be assigned a value in the state from which the heuristic is computed, can be seen as equivalent to *evidence* in the setting of Bayesian networks, which specifies an observed instantiation of a subset of the nodes.

The version of recursive conditioning resulting from these modifications is shown in Figure 6.5. Since the choice variable graphs that we consider are typically small and the time taken by computing the value of a heuristic at each state is the largest factor in heuristic search planners' runtime, we use full caching of computed costs for each subproblem. We construct the dtree used in the algorithm from the elimination ordering suggested by the greedy *min-degree* heuristic, which orders the nodes last to first in increasing order of their degree. Finally, the values that choice variables have in the state from which the heuristic is being computed which therefore cannot be changed, as well as the single values of each of the goal choice variables, are given to the algorithm as evidence.

Proposition 6.8 (Correctness of RC-h)**.** *When the choice variable graph G_C is acyclic, the RC-h algorithm computes the values of the choice variable decomposition heuristic h^{cd} (Equation 6.3).*

```
**Input**: A dtree node $D$
**Input**: A set of choice variables $C$
**Input**: An assignment $\mathbf{v}$ to a subset of $C$
**Input**: A base heuristic function $h$
**Output**: A heuristic estimate $h^{cd} \in \mathbb{R}_0^+$

function RC-h $(D, \mathbf{v})$ begin
    if $D$ is a leaf node then
        $C_i \leftarrow$ the choice variable associated with $D$
        return $h^{\mathbf{v}}(C_i = \mathbf{v}[C_i] \mid Par(C_i))$
    else if $cache_D[\boldsymbol{v}^{context(D)}] \neq undefined$ then
        return $cache_D[\mathbf{v}^{context(D)}]$
    else
        $h^{cd} \leftarrow \infty$
        for $\boldsymbol{c} \in \phi(cutset(D))$ do
            $h^{cd} \leftarrow \min(h^{cd}, \mathtt{RC\text{-}h}(D^l, \mathbf{v} \cup \mathbf{c}) + \mathtt{RC\text{-}h}(D^r, \mathbf{v} \cup \mathbf{c}))$
        $cache_D[\mathbf{v}^{context(D)}] \leftarrow h^{cd}$
        return $h^{cd}$
end
```

Figure 6.5: Recursive conditioning for calculating h^{cd} in a planning problem with choice variables.

Proof. It can be shown by induction on the depth of the dtree that the value calculated by recursive conditioning at each node n of the dtree is

$$f_{\mathbf{v} \in \phi(vars(n))}\ g_{n' \in leaves(n)}\ value(n', \mathbf{v})$$

where f is the function used to aggregate results from different instantiations of the set of variables of the node n, $vars(n)$, g is the function used to aggregate the results of two subtrees for a given instantiation, and $value(n', \mathbf{v})$ is the value returned from a leaf node n' given instantiation $\mathbf{v}$. When $f = \min$, $g = \sum$, and $value(n', \mathbf{v}) = h(C_{n'} = \mathbf{v}[C_{n'}] \mid Par(C_{n'}))$, the result computed by the algorithm is exactly the definition of Equation 6.3. □

Proposition 6.9 (Complexity of RC-h)**.** *The number of calls made to the underlying heuristic estimator h by the RC-h algorithm is $O(n^w)$, where n is the number of nodes in the choice variable graph G_C and w is the width of the dtree used.*

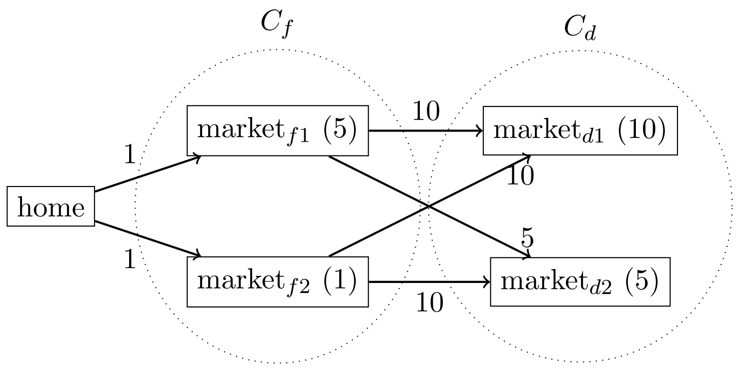


Figure 6.6: A problem in which food and drink must be bought. Food is available at *market*$_{f1}$ and *market*$_{f2}$, while drink is available at *market*$_{d1}$ and *market*$_{d2}$. Numbers in parenthesis give the cost of each good at the specified market, and values next to each edge show the transportation cost for the edge.

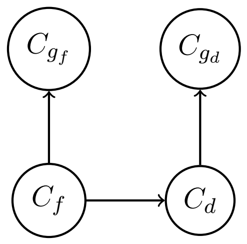
Proof. Direct from results concerning the complexity of recursive conditioning for computing the probability of evidence (Darwiche, 2001). □

We now give a short example of how the recursive conditioning algorithm is used to calculate the values of h^{cd}. We consider a problem of the type discussed above, in which an agent must buy two different types of goods, food and drink, and two different markets are available selling each (Figure 6.6). The resulting choice variable graph then constitutes a tree, with the choice of the market at which to buy food not being dependent on any choice variable, the choice of at which market to buy drink being dependent only on at which market food is bought, and the choice variables for the two goals *have-food* and *have-drink* being dependent only on the market chosen for each type of good. This choice variable graph and a dtree for the graph resulting from the min-degree ordering are shown in Figure 6.7. The RC-h algorithm begins at the root node of the dtree, and must enumerate the possible instantiations of its cutset $\{C_f\}$. *market*$_{f1}$ is selected as the first value, and the recursion proceeds with a call to the right subtree. This is an internal node with an empty cutset, so no further variables are instantiated. The algorithm than recurses to the right child, which is a leaf node in which the base heuristic is used to estimate the cost of the goal *have-food* from the initial state $\{home, market_{f1}\}$, with actions adding other values of the choice variable C_f disallowed. There is a single possible plan for this component which consists of buying food at $market_{f1}$, and this plan has cost 5, which is returned to the parent node. The algorithm then proceeds to evaluate the cost of the left node, which is the cost of making $market_{f1}$ true from the initial state $\{home\}$, 1. Since the cutset of the node is empty, no further instantiations are required and the value $5 + 1 = 6$ is returned to the root node. To estimate the cost of the left subtree of the root, the algorithm now calls itself recursively on the node with cutset $\{C_d\}$, instantiating it first to $market_{d1}$. In the call to the right child, the cost of setting $market_{d1}$ from initial state $\{home, market_{f1}\}$ is evaluated, to give cost 10, and in the left child, the cost of buying drink at $market_{d1}$ is evaluated, to give cost 10. The returned values are summed in the internal node with cutset C_d to give cost 20. The value resulting from $C_d = market_{d2}$ is calculated similarly, giving cost $5 + 5 = 10$, which is lower than the previous instantiation, and therefore returned to the root node, in which it is summed with the cost calculated for the right child to give a final cost of $10 + 6 = 16$ for the instantiation $C_f = market_{f1}$. The cost of choosing $C_f = market_{f2}$ is computed similarly and turns out to be 17, which is higher than the previous estimate, so 16 is returned as the final value.

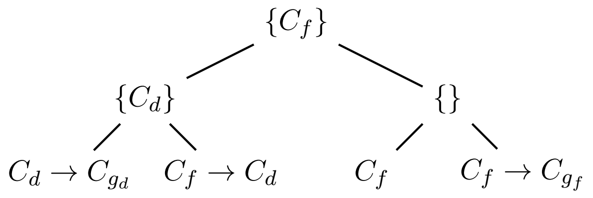
Note that on this problem the optimal delete relaxation plan would consist of moving to $market_{f2}$ and buying food there, then moving from *home* to $market_{f1}$ and from there to $market_{d2}$ to buy drinks there, for a total cost of $1 + 1 + 1 + 5 + 5 = 13$.

Computation

The recursive conditioning algorithm avoids repeated computations by caching at each node the heuristic value computed for a given instantiation of its context. However, since it is designed for the setting of Bayesian nets in which it is only run once to obtain the likelihood of given evidence, it tells us nothing about whether values can be cached to avoid repeated computations



(a) The choice variable graph for the two markets problem.



(b) A dtree for the choice variable graph resulting from the min-degree elimination ordering $\langle C_{g_d}, C_{g_f}, C_d, C_f \rangle$, with cutsets shown for internal nodes.

Figure 6.7: Two markets problem. The families for the choice variable graph, and therefore the leaf nodes of the dtree, consist of $C_f, C_f \to C_d, C_f \to C_{g_f}, C_d \to C_{g_d}$

in different states from which the value of h^{cd} is computed. We observe that the heuristic values $h^{\mathbf{v}}(C_i = \mathbf{v}[C_i] \mid \mathit{Par}(C_i))$ do not change from state to state for components in which there is no non-choice fluent that is conditionally relevant to C_i given $\mathit{Par}(C_i)$. These values can then be cached in each leaf node of the dtree the first time they are computed, and reused in later heuristic computations. When the number of choice variables in each component is small, this can substantially improve the cost of computing the heuristic.

This caching scheme could be further extended to take into account in each component the values of the non-choice fluents conditionally relevant to the target choice variable of the component in the state from which the heuristic is computed. We have not experimented with this approach as the size of the cache at each leaf node would be unbounded, yet it may be possible in certain domains to obtain an increase in performance by caching values only in components for which the number of conditionally relevant non-choice fluents falls below some bound.

6.3 Domains

In this section we present a number of problems that can be encoded using the choice variables formalism, and discuss some non-intuitive features of the encoding that are required for the heuristic to give optimal values when the base heuristic h is taken to be the optimal heuristic h^*.

Minimum Cost SAT

As a canonical example of a constraint satisfaction problem, we consider a version of the boolean satisfiability problem with three literals per clause in which a satisfying assignment with minimum cost must be found (MCSAT) (Li, 2004), and show how to encode it so that the heuristic values found by h^{cd} are optimal. An MCSAT problem consists of a logical formula in conjunctive normal form (CNF) in which each of the conjuncts is associated with a weight, and the objective is to find an assignment such that the total weight of the violated clauses is minimized. The natural choice of choice variables for the problem then consists of the sets $\{x_i, \neg x_i\}$ for each problem variable x_i.

An acyclic interaction graph for the variables of an MCSAT problem can be obtained by imposing an ordering over the set of variables of the problem, and adding a directed edge between two nodes x_i, x_j whenever x_j appears as the highest-ranked literal in a clause in which x_i also appears. This suggests a planning encoding in which variables which are not highest-ranked in any clause can be set freely, while operators setting the values of highest-ranked variables have preconditions that ensure that the clauses for which they are highest-ranked are already satisfied, if the assignment made by the operator itself does not satisfy the clause. Logically, the operators of the problem proceed from the following equivalence:

$$(x_1 \vee x_2 \vee x_3) \iff (\neg(x_1 \vee x_2) \implies x_3)$$

This implies that to set the value of x_3 to false, either x_1 or x_2 must already be true. We encode this set of relationships with two operators for each variable in the problem, one of which sets the variable to true and the other to false. The operator *set-x_i-true* then has as a precondition the conjunction of all of the clauses in which it appears in negated form as the highest-ranked variable, while the operator *set-x_i-false* has as a precondition those in which it appears positively.

As an example, consider the following SAT formula:

$$(x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee \neg x_4)$$

The *set* actions for variable x_4, the only variable that is highest-ranked in any of the clauses, are the following:

$$\begin{aligned} \textit{set-}x_4\textit{-false} :& \langle \{\textit{unset}(x_4), (x_1 \vee x_2)\}, \{\neg x_4, \neg \textit{unset}(x_4), \textit{set}(x_4)\}\rangle \\ \textit{set-}x_4\textit{-true} :& \langle \{\textit{unset}(x_4), (x_1 \vee x_3)\}, \{x_4, \neg \textit{unset}(x_4)\}, \textit{set}(x_4)\}\rangle \end{aligned}$$

while all other variables x_i can be set to true or false with the single precondition $\textit{unset}(x_i)$. The goals of the problem are then $\textit{set}(x_i)$ for all the variables of the problem.

Though this encoding is simple and intuitive, it does not fulfill the conditions stated in Proposition 6.7, as each of the *set-x_i* actions has non-zero cost and is relevant to both the choice variable C_i, since it adds one of its values, and to the goal choice variable representing that a value for the variable has been set, since it adds the fluent $\textit{set}(x_i)$. To get around this, we introduce a second set of actions *finalize-x_i* for each variable x_i. These actions have zero cost and the disjunctive precondition that x_i has already been set to true or false. The $\textit{set}(x_i)$ fluents in the goal are then replaced with the fluents $\textit{finalized}(x_i)$ added by these operators, ensuring that each of the *set-x_i* operators are relevant only to the choice variable to which they set a value, and that each *finalize-x_i* operator is relevant only to the associated goal choice variable C_{g_i}.

Proposition 6.10 (Optimality of h^{cd} on MCSAT)**.** *Given the encoding described above, when* $h = h^*$, $h^{cd} = h^*$.

Proof. The choice variables graph G_C for the encoding described above constitutes a directed acyclic graph in which each of the $set - x_i$ actions is conditionally relevant only to the choice variable C_i corresponding to x_i. Optimality then follows from Proposition 6.7. □

Proposition 6.11 (Time complexity of h^{cd} on MCSAT)**.** *Given an elimination ordering of width* w *for the choice variables graph* C_G*, the number of calls to the underlying heuristic* h *made by the RC-h procedure is* $O(n^w)$.

Proof. Given an elimination ordering of width w, the width of the constructed dtree will be $\leq w$. The total number of recursive calls made by RC-h is then $O(n^w)$, where n is the number of leaf nodes in the dtree, or equivalently, the number of variables in the problem. The number of calls to the underlying heuristic h is then also $O(n^w)$. □

As delete relaxation heuristics based on the independence assumption can be computed in time polynomial in the size of the problem, the runtime of the algorithm when using such a heuristic is then also bounded by $O(n^w)$, which is the complexity bound of the MCSAT problem. Note that while the number of calls to the base heuristic is exponential in the width, the problem which must be solved in each call is simply choosing a value for a variable x_i given values for each of its parents $Par(x_i)$ in G_C, which are those variables which appear together with x_i in some clause in which x_i is the highest-ranked fluent.

Hidden Markov Models

A hidden Markov model (HMM) is a dynamic Bayesian process defined by $H = \langle S, O, T, I, E \rangle$, where S is a set of states, O is a set of observations, T is the transition probability function, with each $T(s' \mid s)$ describing the probability of transitioning to state s' given that the current state is s, I is the initial state probability function, with each $I(s)$ giving the probability that the initial state of the process is s, and E is the evidence probability function, with each $E(o|s)$ giving the probability of observation o given that the current state of the process is s. Given a sequence of observations $\langle o_1, \ldots, o_t \rangle$, the *most probable explanation* (MPE) problem is that of computing the sequence of states $\langle s_0, \ldots, s_t \rangle$ which has the highest probability of generating the observed sequence, where the probability of a sequence of states' generating some sequence of observations is given by $I(s_0) \prod_{i=1}^{t} T(s_i \mid s_{i-1}) E(o_i \mid s_i)$.

The MPE problem can be encoded as a planning problem $\Pi = \langle F, I, O, G \rangle$ as follows:

- $F = \{\mathit{state}_i(s), \mathit{obs}_i(o), \mathit{begin}\}$, for $i = 0, \ldots, t$, $s \in S$, and $o \in O$
- $I = \{\mathit{begin}\}$
- $O = \{\mathit{set\text{-}initial\text{-}state}(s), \mathit{transition}_i(s, s'), \mathit{emit}_i(s, o)\}$
- $G = \{\mathit{obs}_j(o_j)\}$ for $j = 1 \ldots t$

and each operator $o = \langle \mathit{Pre}(o), \mathit{Eff}(o) \rangle$ is described by

- $\mathit{set\text{-}initial\text{-}state}(s) = \langle \{\mathit{begin}\}, \{\mathit{state}_0(s), \neg \mathit{begin}\} \rangle$, with $\mathit{cost} = -\mathit{ln}(I(s))$
- $\mathit{transition}_i(s, s') = \langle \{\mathit{state}_i(s)\}, \{\mathit{state}_{i+1}(s'), \neg \mathit{state}_i(s)\} \rangle$, with $\mathit{cost} = -\mathit{ln}(T(s'|s))$
- $\mathit{emit}_i(s, o) = \langle \{\mathit{state}_i(s)\}, \{\mathit{obs}_i(o)\} \rangle$, with $\mathit{cost} = -\mathit{ln}(E(o|s))$

The optimal plans for Π then encode the most probable explanations for the sequence of observations $\langle o_1, \ldots, o_j \rangle$. The use of logarithmic costs in place of probabilities is a standard method in probabilistic applications, where it is used to avoid handling small floating point numbers that cannot be represented exactly. It is especially useful in the planning setting of additive costs. The choice variables $C_0, \ldots, C_t$ for this problem can be chosen

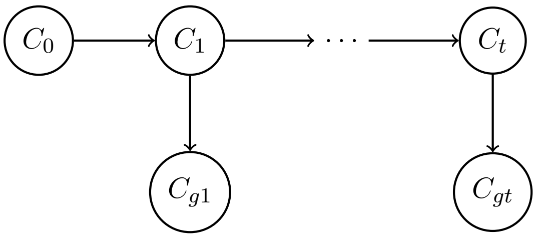


Figure 6.8: The choice variable graph of width 1 that results from the encoding of the MPE problem described below.

as the sets of fluents $C_i = \cup_{s \in S}\{state_i(s)\}$, resulting in a choice variables graph G_c of width 1, in which the state at each timestep i depends only on the state at the timestep $i-1$, and the observation at timestep i depends only on the state at timestep i. (Figure 6.8).

Proposition 6.12 (Optimality of h^{cd} on MPE for HMMs)**.** *Given the encoding described above, when $h = h^*$, $h^{cd} = h^*$.*

Proof. The choice variables graph G_C for the encoding described above constitutes a directed tree in which the *set-initial-state* operator is relevant only to C_0, each of the $transition_i$ operators is relevant only to C_{i+1}, and each of the $emit_i$ operators is relevant only to the goal choice variables, each of whose domains consists of $\{obs_i(o_i)\}$. Optimality then follows from Proposition 6.7. □

Proposition 6.13 (Time complexity of h^{cd} on MPE for HMMs)**.** *Given the optimal elimination ordering of width 1, the number of calls made by RC-h to the underlying heuristic during each computation is $O(n)$.*

Proof. Direct from Proposition 6.9. □

We note that the use of the min-degree heuristic to obtain an elimination ordering in a tree will always result in an optimal elimination ordering of width 1.

6.4 Experimental Results

We compare the performance of the choice variable heuristic h^{cd} on the domains described above to that of a standard delete relaxation heuristic, the cost of the relaxed plan obtained from the additive heuristic h^{add} (Chapter 3). We use the same heuristic within the framework of h^{cd} to obtain the heuristic estimates for the cost of each component $h^{\mathbf{v}}(C_i = \mathbf{v}[C_i] \mid Par(C_i))$. The heuristics are used in the context of a greedy best-first search with delayed evaluation, using a second open queue for states resulting from helpful actions. Helpful actions for h^{cd} are obtained as those actions which are helpful in any one of the problem components according to the underlying heuristic and which can be applied in the current state. In general, the performance of h^{cd} is heavily dependent on the width of the choice variable graph G_C for the problem.

All experiments discussed below were run on Xeon Woodcrest computers with clock speeds of 2.33 GHz, using a 2GB memory limit and a time cutoff of 1800 seconds when appropriate.

Minimum Cost SAT

We generated MCSAT problems in the planning encoding described above, with the number of clauses chosen to be the number of variables times 4.3, a ratio which has been shown to produce problems for which satisfiability testing is hard (Mitchell et al., 1992). The number of variables in the problems ranged from 5 to 34, and the cost of making each variable true or false was set to be 5 and 1, respectively. Table 6.1 shows each heuristics estimate of the cost of the problem, i.e. the result of evaluating the heuristics on the initial state. The choice variable heuristic h^{cd} computes the problem's optimal cost, while the additive heuristic h^{add} produces both overestimates and underestimates. Problems for which no satisfying assignment is possible are reported by h^{cd} to have infinite cost with a single heuristic evaluation, while search with h^{add} requires a large number of node expansions before this condition is proved. In our evaluation, we therefore included only problems with satisfying assignments.

Problem	h^{add}	h^{cd}	Problem	h^{add}	h^{cd}
sat-5-21	14	17	sat-20-86	81	52
sat-6-25	16	26	sat-21-90	68	41
sat-7-30	27	15	sat-22-94	87	62
sat-8-34	26	16	sat-23-98	106	55
sat-9-38	46	29	sat-24-103	104	64
sat-10-43	24	22	sat-25-107	92	77
sat-11-47	36	31	sat-26-111	109	82
sat-12-51	48	20	sat-27-116	106	63
sat-13-55	42	37	sat-28-120	90	56
sat-14-60	52	26	sat-29-124	120	89
sat-15-64	62	31	sat-30-129	111	82
sat-16-68	51	36	sat-31-133	130	99
sat-17-73	50	41	sat-32-137	137	80
sat-18-77	88	46	sat-33-141	158	101
sat-19-81	66	55	sat-34-146	139	94

Table 6.1: Heuristic estimates for the initial states of MCSAT problems. *sat-x-y* indicates a problem with x variables and y clauses.

When used in the framework of a greedy best-first search with delayed evaluation, h^{cd} is able to solve the smaller problems that we consider with much fewer heuristic evaluations than h^{add}. In fact, since the heuristic is optimal for these problems, the number of heuristic evaluations is roughly linear in the number of variables for the problem, and therefore in the length of the plan, since all plans for a given instance of the domain have the same length (Figure 6.9). However, for the ratio of clauses to variables that we consider, the interaction graph of the variables is usually clique-like, resulting in a choice variable graph, and therefore a dtree, whose width is unbounded and typically close to the number of variables in the problem. The computation of h^{cd} therefore rapidly becomes infeasible, and h^{add} is able to solve many more problems, 29 of the 30 compared to 20. The higher informativeness of h^{cd} does not pay off in terms of plan cost either, with the eventual plans found by greedy best first having roughly equal cost for both heuristics, with h^{cd} finding plans of only slightly less cost for the larger problems it is able

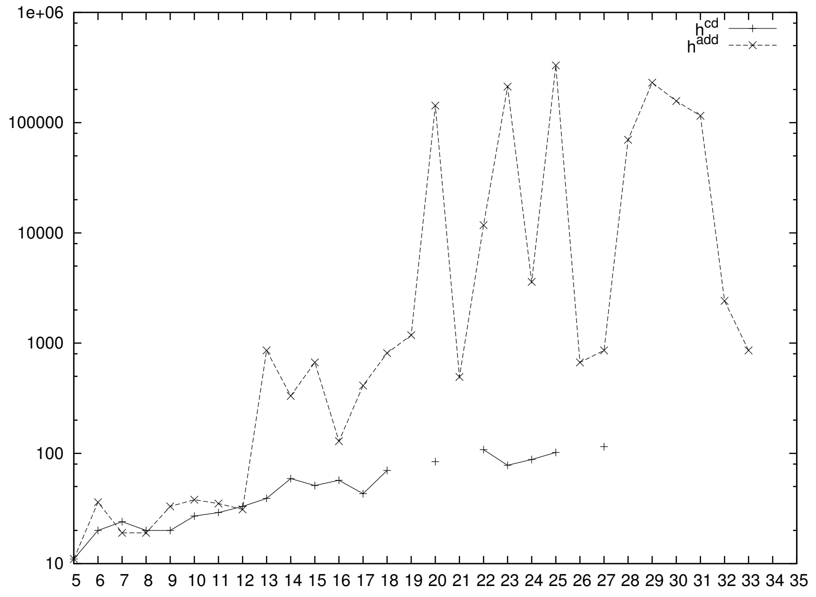


Figure 6.9: Node expansions on the MCSAT domain.

to solve.

Hidden Markov Models

We generated most probable explanation problems on hidden Markov models using the planning encoding described above, keeping the number of states and the number of different possible observations in the model fixed at 15, and varying the length of the observed sequence to be explained. Table 6.2 shows the results of evaluating both heuristics for the initial states of the problems. Recall that the costs used in the problem encoding are the natural logarithms of the probabilities in the model; linear differences between the heuristic estimates therefore reflect order-of-magnitude differences in the estimated likelihood of the observed sequence. The h^{cd} heuristic finds optimal explanations for the sequence of observations that are several orders of magnitude more likely than those found by h^{add}, with the effect becoming more pronounced as the length of the observation sequence used is increased. In particular, for the largest problem

Problem	h^{add}	h^{cd}	Problem	h^{add}	h^{cd}
hmm-15-50	261.8	106.3	hmm-15-200	1387.5	484.5
hmm-15-60	521.7	122.1	hmm-15-210	1175.1	436.8
hmm-15-70	472.4	115.0	hmm-15-220	1545.1	520.3
hmm-15-80	735.9	191.0	hmm-15-230	3091.8	608.8
hmm-15-90	810.8	160.8	hmm-15-240	1207.8	605.8
hmm-15-100	724.3	239.4	hmm-15-250	2544.3	658.8
hmm-15-110	761.6	267.8	hmm-15-260	2819.8	618.5
hmm-15-120	549.7	302.3	hmm-15-270	1758.3	581.4
hmm-15-130	1184.7	258.8	hmm-15-280	1359.3	580.3
hmm-15-140	955.7	319.5	hmm-15-290	1587.6	728.5
hmm-15-150	1434.6	348.9	hmm-15-300	3089.3	717.3
hmm-15-160	1380.0	320.8	hmm-15-310	2170.5	810.8
hmm-15-170	1543.5	269.2	hmm-15-320	4532.7	804.7
hmm-15-180	1154.6	397.4	hmm-15-330	3274.1	754.2
hmm-15-190	1096.0	423.4	hmm-15-340	3111.0	860.0

Table 6.2: Heuristic estimates for the initial states of most probable explanation problems on hidden Markov models. *hmm-x-y* indicates a problem with x states and possible observations, and an observed sequence length of y.

used, in which the length of the observation sequence is 340, the optimal estimate given by the h^{cd} heuristic shows that the likelihood of the sequence is more than 100 orders of magnitude greater than what the additive heuristic's estimate suggests.

When the two heuristics are used in greedy best-first search, search with h^{cd} is able to scale up to much larger sequences of observations than h^{add}, solving all 30 of the problems, compared to search with h^{add} that solves only 21. This is due to the fact that the choice variable graph for the problem is a tree with the structure seen in Figure 6.8, and its width is therefore 1, independently of the length of the observed sequence in the problem. The min-degree heuristic is always able to find an optimal elimination ordering for trees, resulting in a dtree with width 1 as well. The complexity of computing h^{cd} then grows linearly in the size of the problem, and the optimal values computed allow it to solve problems with a roughly linear number of heuristic evaluations as well (Figure 6.10). The heuristic is also benefi-

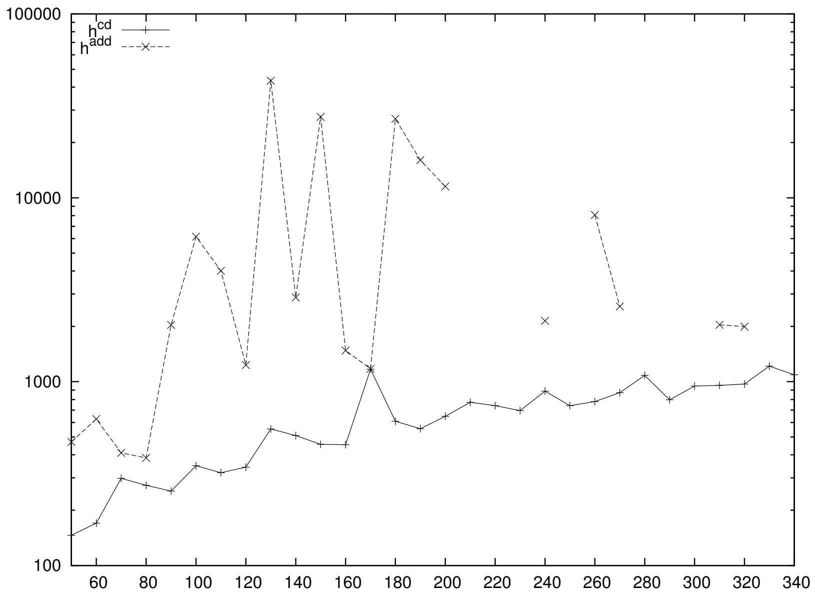


Figure 6.10: Node expansions on the HMM domain.

cial in terms of the cost of the plans that are found, with h^{cd} finding lower cost plans for all of the instances solved with both heuristics, and the difference in cost growing with the size of the problem (Figure 6.11)

6.5 Related Work

As stated above, our work contains several similarities to factored planning approaches, first proposed by Amir and Engelhardt (2003). Similarly to the technique presented here, factored planning relies on decomposing a planning problem into several subproblems, or components. These subproblems are then solved with a standard planner, which finds plans that mix operators from within the component and pseudo-actions that consist of some other component changing the value of a variable shared between the two components. The number of times d in which a different component changes the value of an internal variable in this manner is a parameter of the planning

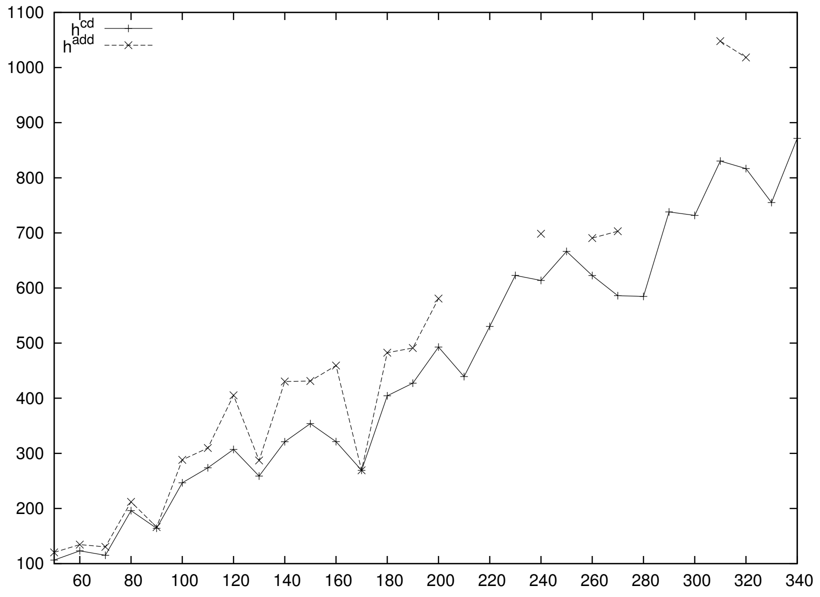


Figure 6.11: Plan costs on the HMM domain.

algorithm. Brafman and Domshlak (2006) later showed that the minimum number of times that some variable must change its value in a plan is a key parameter in determining the complexity of planning problems. Our approach differs from these in that it aims to define a heuristic estimator for the problem rather than lay out an algorithm that produces an explicit solution. Furthermore, we focus on problems in which the variables that are shared between components are restricted to having their values changed at most once. However, since our methods are aimed at producing a heuristic estimate for the problem and not an explicit solution, they can easily be extended to this setting with the single-change assumption constituting a relaxation of the problem similar to the delete relaxation.

6.6 Conclusions

We have introduced a new type of invariant in planning that we call a choice variable. A choice variable is a multivalued variable whose values can be set at most once by any plan. By reasoning by cases about different assignments to these variables and excluding the operators that violate the assignments, heuristics that go beyond the delete relaxation in their estimates can be obtained. Furthermore, this reasoning can be done efficiently by adapting techniques previously proposed in the context of graphical models to the planning setting, with the complexity of the heuristic computation depending on the well-studied width parameter of the graph that describes the interaction between choice variables. More than a device for improving heuristic estimates on already existing domains, choice variables can also be seen as a modelling tool for embedding common problems in computer science that have graphical model-like structure into planning problems, extending the range of problems for which planning algorithms can be effectively used.

Part IV

Extending the Scope of Classical Planning

Chapter 7

Compiling Soft Goals Away

In this chapter we describe the problem of planning with soft goals and discuss some previous approaches to the problem. We then show how this type of problem can be compiled into a standard planning problem with only hard goals and action costs, and discuss the importance of cost-sensitive and non-independence-assumption-based heuristics in solving the compiled problems. Finally, we evaluate the performance of off-the-shelf satisficing and optimal classical planners on some compiled problems and compare this to the performance of off-the-shelf planners that deal with soft goals using dedicated algorithms.

7.1 Introduction

Planning with soft goals extends the classical model of planning with a simple model of preferences over the final state that is reached by the plan. This is done by associating with each fluent $p \in F$ a *utility*, with a utility of 0 indicating that plans are not preferred on the basis of whether they make p true or not, and a positive value indicating that between two plans of equal cost, one of which results in a final state in which p is true and the other in which in a final state in which p is false, the plan making

p true is preferred. The magnitude of the utility represents how much extra cost should be accepted by the planner in order to make p true: if plan π makes p true and has cost c, and plan π' does not and has cost c', π is preferred to π' if the utility conferred by achieving p is greater than the extra cost it incurs, *i.e.* when $c - c' < utility(p)$.

Formally, soft goals are described in STRIPS with an extension to the STRIPS planning problem with costs (see Definition 1.6):

Definition 7.1 (STRIPS with soft goals). *A* STRIPS *problem with action costs and soft goals* $P_u = \langle F, I, O, G, c, u \rangle$ *consists of:*

- *A* STRIPS *problem with costs* $P = \langle F, I, O, G, c \rangle$
- *A utility function* $u : F \mapsto \mathbb{R}_0^+$ *that assigns to each fluent in the problem a non-negative utility.*

In the rest of this chapter, we refer to the subset of fluents associated with a positive utility as the soft goals of the problem, and denote them with $SG(P) = \{f \in F \mid u(f) > 0\}$.

In planning problems with costs but no soft goals, the cost of a plan is defined to be the summed cost of the operators that appear in the plan (Definition 1.4). For planning problems with soft goals, whether a fluent with non-zero utility is made true or not in the final state resulting from the execution of the plan must also be taken into account. In the following, we denote this condition, $p \in s_I[\pi]$, with the notation $\pi \models p$. The *utility* of a plan $u(\pi)$ is then given by the following expression:

$$u(\pi) = \sum_{p:\pi \models p} u(p) - cost(\pi) \tag{7.1}$$

A plan π for a problem with soft goals P_u is optimal when no other plan π' has utility $u(\pi')$ greater than $u(\pi)$. The utility of an optimal plan for a problem that has no hard goals, *i.e.* such that $|G| = 0$, is never negative, as the empty plan has non-negative utility and zero cost.

We now present a compilation technique that given a planning problem with soft goals transforms it into a problem with only hard goals and action costs. This problem can then be solved by an off-the-shelf classical planner with no enhancements, taking advantage of cost-sensitive heuristic planning techniques.

7.2 Compiling Soft Goals Away

Given a STRIPS problem P with soft goals, an equivalent problem P' with action costs and *no soft goals* can be defined whose plans encode corresponding plans for P.

In the following, we write actions as tuples of the form $o = \langle \mathit{Pre}(o), \mathit{Eff}(o) \rangle$, where the effects can be positive (Adds) or negative (Deletes). We assume that for each soft goal fluent p, P also contains a fluent $\overline{p}$ representing its negation. These can be introduced in the standard way, adding $\overline{p}$ to the initial state if p is not initially true, and including $\overline{p}$ in the *Add* and *Delete* lists of all actions deleting or adding p respectively (Gazen and Knoblock, 1997; Nebel, 2000). The problem P' with action costs and no soft goals that is equivalent to the problem P with soft goals can then be obtained by the following transformation:

Definition 7.2. *For a STRIPS problem with action costs and soft goals $P = \langle F, I, O, G, c, u \rangle$, the compiled STRIPS problem with action costs is $P' = \langle F', I', O', G', c' \rangle$ with*

- $F' = F \cup S'(P) \cup \overline{S}(P) \cup \{\text{NORMAL-MODE}, \text{END-MODE}\}$
- $I' = I \cup \overline{S}(P) \cup \{\text{NORMAL-MODE}\}$
- $G' = G \cup S'(P)$
- $O' = O'' \cup \{\text{COLLECT}(p), \text{FORGO}(p) \mid p \in SG(P)\} \cup \{\text{END}\}$
- $c'(o) = \begin{cases} c(o) & \textit{if} \quad o \in O'' \\ u(p) & \textit{if} \quad o = \text{FORGO}(p) \\ 0 & \textit{if} \quad o = \text{COLLECT}(p), \text{END} \end{cases}$

where

- $SG(P) = \{p \mid (p \in F) \wedge (u(p) > 0)\}$

- $S'(P) = \{p' \mid p \in SG(P)\}$
- $\overline{S}(P) = \{\overline{p'} \mid p' \in S'(P)\}$
- $\textsc{end} = \langle\{\textsc{normal-mode}\}, \{\textsc{end-mode}, \neg\textsc{normal-mode}\}\rangle$
- $\textsc{collect}(p) = \langle\{\textsc{end-mode}, p, \overline{p'}\}, \{p', \neg\overline{p'}\}\rangle$
- $\textsc{forgo}(p) = \langle\{\textsc{end-mode}, \overline{p}, \overline{p'}\}, \{p', \neg\overline{p'}\}\rangle$
- $O'' = \{\langle Pre(o) \cup \{\textsc{normal-mode}\}, Eff(o)\rangle \mid o \in O\}$

For each soft goal p in P, the transformation adds a dummy hard goal p' in P' that can be achieved in two ways: with the action COLLECT(p) that has cost 0 but requires p to be true, or with the action FORGO(p) that has cost equal to the utility of p yet can be performed when p is false, or equivalently when $\overline{p}$ is true. These two actions can be used only after the *end* action that makes the fluent END-MODE true, while the actions from the original problem P can be used only when the fluent NORMAL-MODE is true prior to the execution of the END action. Moreover, exactly one of $\{\textsc{collect}(p), \textsc{forgo}(p)\}$ can appear for each soft goal p in the plan, as both delete their shared precondition $\overline{p'}$, which no action makes true. As there is no way to make the fluent NORMAL-MODE true again after it is deleted by the *end* action, all plans π' for P' have the form $\pi' = \langle \pi, end, \pi'' \rangle$, where π is a plan for P and π'' is a sequence of $|S'(P)|$ COLLECT(p) and FORGO(p) actions in any order, the former appearing when $\pi \models p$, and the latter otherwise.

The two problems P and P' are equivalent in the sense that there is a correspondence between the plans for P and P', and corresponding plans are ranked in the same way. More specifically, for any plan π for P, there is a plan π' for P' that extends π with the END action and a set of COLLECT and FORGO actions, and this plan has cost $c(\pi') = -u(\pi) + \alpha$, where α is a constant that is independent of both π and π'. Finding an optimal (maximum utility) plan π for P is therefore equivalent to finding an optimal (minimum cost) plan π' for P'.

Proposition 7.3 (Correspondence between plans)**.** *For an applicable action sequence π in P, let an* extension π' *of π denote* *any sequence obtained by appending to π the end action followed by some permutation of the actions* COLLECT(p) *and* FORGO(p) *for all $p \in SG(P)$, when $\pi \models p$ and $\pi \not\models p$ respectively. Then*

$$\pi \textit{ is a plan for } P \iff \pi' \textit{ is a plan for } P'$$

Proof. ($\Rightarrow$) The new actions in P' do not delete any $p \in F$, so any hard goal achieved by π will remain true in the final state reached by π', and we have that $\pi' \models G$. For all $p \in F$ such that $u(p) > 0$, either $\pi \models p$ or $\pi \not\models p$. In the first case, p' is achieved by COLLECT(p), in the second, by FORGO(p), therefore $\pi' \models S'(P)$. Since $G' = G \cup S'(P)$, we have that $\pi' \models G'$.

($\Leftarrow$) If π' is a plan for P', then all hard goals G in P must be made true by π' before the END action, as after this action only COLLECT and FORGO actions can be applied and these can not make any $p \in F$ true. The plan obtained by removing the END action and all COLLECT and FORGO actions must therefore achieve G and thus is a valid plan for P. □

Proposition 7.4 (Correspondence between utilities and costs)**.** *Let π_1 and π_2 be two plans for P, and let π'_1 and π'_2 be extensions of π_1 and π_2 respectively. Then,*

$$u(\pi_1) > u(\pi_2) \iff c(\pi'_1) < c(\pi'_2)$$

Proof. Let π be a plan for P and π' an extension of π. We demonstrate that $c(\pi') = -u(\pi) + \sum_{p \in SG(P)} u(p)$. Since the summation in this expression is a constant for a given problem P, the assertion follows directly:

$$\begin{aligned}
c(\pi') &= c(\pi) + c'(\textsc{end}) + \\
&\quad \sum_{\textsc{forgo}(p)\in\pi'} c'(\textsc{forgo}(p)) + \\
&\quad \sum_{\textsc{collect}(p)\in\pi'} c'(\textsc{collect}(p)) \\
&= c(\pi) + \sum_{\textsc{forgo}(p)\in\pi'} c'(\textsc{forgo}(p)) \\
&= c(\pi) + \sum_{p:\pi\not\models p} u(p) \\
&= c(\pi) + \sum_{p\in SG(P)} u(p) - \sum_{p:\pi\models p} u(p) \\
&= -u(\pi) + \sum_{p\in SG(P)} u(p)
\end{aligned}$$

□

Proposition 7.5 (Equivalence)**.** *Let π be a plan for P, and π' be a plan for P' that extends π. Then,*

$$\pi \textit{ is an optimal plan for } P \iff \pi' \textit{ is an optimal plan for } P'$$

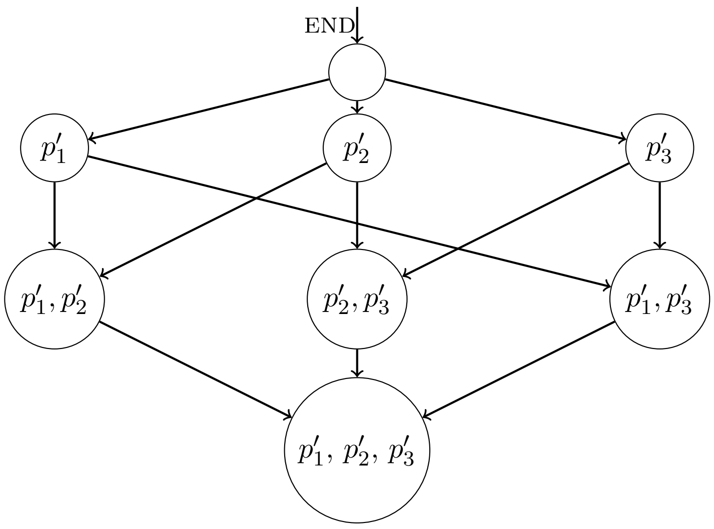
Proof. Direct from the two propositions above. □

Optimizing the Compilation

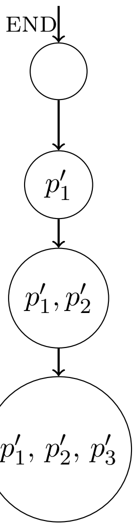
While the problem resulting from the compilation is equivalent to the original problem in the sense shown above, it increases the state space of the problem by a large factor, which can have a hugely detrimental effect on planners' performance. Though the state in which the END action is applied uniquely determines which of $\{\textsc{collect}(p), \textsc{forgo}(p)\}$ can be applied for each $p \in SG(P)$, it does not specify the order in which they must appear. For a problem with n soft goals, this means that the part of the state space reachable with the END action must contain $|\mathcal{P}(n)|$ different reachable states, in each of which one subset of the n hard goals corresponding to the soft goals of the original problem has been processed (Figure 7.1a). In a problem with no hard goals, the END action can be applied in any state, and this has the overall effect of increasing the size of the state space by a factor of 2^n. However, it turns out that this exponential blowup can be avoided with the simple trick of enforcing a fixed ordering $p_1, \ldots, p_m$ over the set of soft goals in P, preventing the COLLECT and FORGO from being applied in any other order. This is achieved by adding the dummy hard goal p'_i as a precondition of the actions COLLECT(p_{i+1}) and FORGO(p_{i+1}) for $i = 1, \ldots, m-1$. The result is that there is a single possible extension π' of every plan π in P (Figure 7.1b), and the state space of the problem is increased by only a factor of n. This optimization is used in the experiments we report below.

Heuristics and the Soft Goal Compilation

While the compilation technique above permits the solution of problems with soft goals directly by classical planners, it does not tell us anything about whether good quality solutions that take into account both the cost of achieving a soft goal and its associated utility will be found. Indeed, it is easy to see that any planner or heuristic that does not take into account action costs will find very poor quality solutions to compiled problems of this type. This is because for any problem P with n soft goals but no hard goals, the plan $\pi = \langle \text{END}, \text{FORGO}(p_1), \ldots, \text{FORGO}(p_n) \rangle$ will minimize the plan metric $|\pi|$, the number of actions appearing in the plan, as every plan must contain the END action and at least one of $\{\text{COLLECT}(p_i), \text{FORGO}(p_i)\}$ for $i = 1, \ldots, n$. On the other hand, planners and heuristics that take into account action costs will attempt to estimate the cost of plans for each p'_i that have the form $\pi = \langle \pi', \text{END}, \text{COLLECT}(p_i) \rangle$, where π' is a plan achieving the soft goal p_i, and choose to achieve the newly introduced hard goal p'_i with the FORGO(p_i) action only if the cost of achieving p_i is estimated to be greater than the associated utility $u(p_i)$.



(a) State space following END action with unordered soft goals.



(b) State space following END action with ordered soft goals.

Figure 7.1: The soft goal ordering optimization for a problem with 3 soft goals.

7.3 Experimental Results

The formal results above imply that the best plans for a problem P with action costs and soft goals can be computed by looking for the best plans for the compiled problem P' with action costs and no soft goals, to which standard classical planning techniques can be applied. To test the practical value of the transformation, we evaluate the performance of both optimal and satisficing planning techniques for soft goals. Some problems in the test suite contain preferences over conjunctions rather than single fluents. Such preferences are handled with a variant of the approach described above, detailed in Section 7.4.

The results shown in the three columns in Table 7.1 labelled 'Net-benefit optimal planners' are the results as reported by the organizers of the 2008 International Planning Competition (IPC6) (Helmert et al., 2008). All other results were obtained using the same machines and settings as used in the competition: Xeon Woodcrest computers with clock speeds of 2.33 GHz, with a time limit of 30 minutes and a memory limit of 2GB.

In the first set of experiments, we consider the problems used in the *Net Benefit Optimal* (NBO) track of IPC6, in which soft goals are defined in terms of *goal-state preferences* (Gerevini and Long, 2006), and compare the results obtained by the three optimal net-benefit planners with the results obtained by their *Sequential Optimal* (SO) variants on their compilations.[1] The three planners entered in the NBO track of IPC6 were Gamer, Mips-XXL, and HSP^*_P. The SO planners we test on the compiled versions of the NBO problems are the SO versions of Gamer (Edelkamp and Kissmann, 2008) and Mips-XXL (Edelkamp and Jabbar, 2008) and the two SO planners HSP^*_F and HSP^*_0 (Haslum, 2008).[2] These were ranked first, fifth, second, and third, respec-

[1]The compiled problems are currently available at `http://ipc.informatik.uni-freiburg.de/Domains`.

[2]All versions of HSP^* have a bug which may cause suboptimal or invalid solutions to be computed in domains with non-monotonic numeric variables (numeric variables whose values may both increase and decrease) that occur in preconditions of actions or goals (See `http://ipc.informatik.uni-freiburg.de/Planners`). Such variables are present only in the *transport* domain out of all those tested, yet plans computed by HSP^* for both

	NBO			SO			
Domain	G	H_P	M	G	H_F	H_0	M
crewplanning(30)	4	16	8	-	8	**21**	8
elevators (30)	11	5	4	**19**	8	8	3
openstacks (30)	**7**	5	2	6	4	6	1
pegsol (30)	24	0	23	22	**26**	14	22
transport (30)	12	12	9	-	**15**	**15**	9
woodworking (30)	13	11	9	-	10	**14**	7
total	71	49	55	(47)	71	**78**	50

Table 7.1: Coverage for optimal planners: The leftmost three columns give the number of problems solved by each of the planners in the Net Benefit Optimal (NBO) track of IPC6, as reported by the competition organizers (G = Gamer, H_P=HSP^*_P, M = Mips-XXL). The rightmost four columns give the number of *compiled* problems solved by the Sequential Optimal (SO) versions of these planners (G = Gamer, H_F=HSP^*_F, H_0=HSP^*_0, M = Mips-XXL). Dashes indicate that the version of the planner could not be run on that domain.

tively, in the SO track (Helmert et al., 2008). Three out of the six domains from the NBO track of IPC6 involve numeric variables that appear in the preconditions of actions. The SO version of Gamer does not handle numeric variables, and we are therefore unable to run Gamer on such problems. Numeric variables never appear as soft goals and are left untouched by our compilation.

The data in Table 7.1 show that the two HSP^* planners from the SO track run on the compiled problems do as well as, or better than, the best planner from the NBO track run on the original problems with soft goals. The maximum number of solved problems for a domain is higher for the NBO track planners in only a single domain, openstacks (7 vs. 6). In all other domains, SO planners are able to solve a larger number of problems than the maximum number solved by any NBO planner. Considering the

versions of the domain turn out to be valid (as verified by the VAL plan validator, (Howey and Long, 2003)) and optimal in the instances in which they can be checked against the costs of plans computed by other planners.

	NBS			SCS
Domain	S	Y	M	L
elevators (30)	0	0	8	**23**
openstacks (30)	2	0	2	**28**
pegsol (30)	0	5	23	**29**
rovers (20)	8	2	1	**17**
total	10	7	34	**97**

Table 7.2: Coverage and quality for satisficing planners: The entries indicate the number of problems for which the planner generated the best quality plan (S = SGPlan, Y= YochanPS, M=Mips-XXL, L=LAMA). The leftmost three columns are Net Benefit Satisficing (NBS) planners, while LAMA is a Sequential Cost Satisficing (SCS) planner.

performance of the NBO and SO variants of each planner, the compilation benefits most the two versions of the heuristic search planner HSP*, leaving the BDD planners Gamer and Mips-XXL relatively unaffected. Interestingly, HSP$^*_0$ using the compilation ends up solving more problems than Gamer, the winner of the NBO track (78 vs. 71). The drastically better performance of the SO versions of HSP* compared to the net-benefit version is the result of the simple scheme for handling soft goals in the latter, in which optimal plans are computed for each possible subset of soft goals in the problem (roughly), and a change in the search algorithm from IDA* to A*.

In the second set of experiments, we consider the three domains from the NBO track of IPC6 which do not contain numeric variables in the preconditions of actions, and the domain *rovers* from the net-benefit track of IPC5. Domains containing numeric variables in the preconditions of actions are not considered due to the lack of state-of-the-art cost-based planners able to handle them. Domains other than *rovers* from the NB track of IPC5 are not considered as they contain disjunctive, existentially qualified, or universally qualified soft goals which our current implementation does not support. The satisficing net-benefit planners we test on these problems are SGPlan (Hsu and Wah, 2008), the winner of the net benefit track from IPC5, YochanPS (Benton et al., 2009), which received a distinguished performance award in the same competition, and a satisficing variant of MIPS-XXL, which also received a distinguished performance award in that competition and competed in the optimal track of IPC6. We solve the compiled versions of the problems with LAMA, the winner of the sequential satisficing track from IPC6. YochanPS, MIPS-XXL, and LAMA are anytime planners, and the results discussed below refer to the cost of the best plan found by each at the end of the evaluation period of 30 minutes.

Entries in Table 7.2 show the number of problems in each domain for which the plan generated by a planner is the best or only plan produced. We report this data rather than showing graphs of plan utilities as the absolute difference between the quality of plans is not meaningful in itself except when the shortest plans (that ignore costs and/or soft goals) for the problem are significantly more costly. The results show that running a state-of-the-art cost-based planner on the compiled problems yields the best plan in 98 out of the total 110 instances, almost three times the number of instances in which the best-performing native soft goals planner, MIPS-XXL, gives the best plan. Furthermore, in 22 out of the 23 problems for which MIPS-XXL finds the best plan in the *pegsol* domain, LAMA finds a plan with the same quality. The problems in which satisficing net-benefit planners outperform LAMA run on the compiled problems are therefore very few.

These results appear to contradict the results reported by Benton et al. (2009), where the native net-benefit planner, Yochan$^{\mathrm{PS}}$, yields better results than a cost-based planner, Yochan$^{\mathrm{COST}}$, run on problems compiled according to an earlier version of our transformation (Keyder and Geffner, 2007). The discrepancy appears to be the result of the non-informative cost-based heuristic used in Yochan$^{\mathrm{COST}}$, which leads to plans that forgo all soft goals, and the fact that they do not make use of the optimization discussed at the end of Section 7.2, which results in an unnecessary blowup of the state space.

7.4 Extensions

We have shown that it is possible to compile away positive utilities $u(p)$ associated with single fluents p. We show now that this compilation can be extended to deal with positive utilities defined on *formulas over fluents* and to *negative utilities* defined on both single fluents and formulas. Negative utilities stand for conditions to be avoided rather than sought; for example, a utility $u(p \wedge q) = -10$ penalizes a plan that results in a state where both p and q are true with an extra cost of 10. The compilation of soft goals defined on formulas is based on the standard compilation of goal and precondition formulas in classical planning (Gazen and Knoblock, 1997; Nebel, 1999).

A positive utility on a logical formula A can be compiled away by introducing a new fluent p_A that can be achieved at zero cost from any end state where A holds, and by assigning the utility associated with A to p_A. If A is a DNF formula $D_1 \vee \ldots \vee D_n$, it suffices to add n new actions $a_1, \ldots, a_n$ with $a_i = \langle D_i, p_A \rangle$ for $i = 1, \ldots, n$. If A is a CNF formula $C_1 \wedge \ldots \wedge C_n$, a fluent p_i is introduced for each $i = 1, \ldots, n$, along with actions $a_{ij} = \langle C_{ij}, p_i \rangle$ for $j = 1, \ldots, |C_i|$, where C_{ij} stands for the jth fluent of C_i. We also introduce an action $a = \langle \{p_1, \ldots, p_n\}, p_A \rangle$ that allows the addition of fluent p_A in states where A holds. All the newly introduced actions have zero cost, and must be applicable in P' after the actions of the original problem P and before the COLLECT and FORGO actions. The best extensions of any plan π that achieves A in P will then achieve p_A and use the COLLECT action to achieve the hard goal fluent p'_A associated with p_A at zero cost.

A negative utility $u(A) < 0$ on a formula A in DNF or CNF can be compiled away in two steps, by first substituting a positive utility $-u(\neg A)$ on the negation $\neg A$ of A and then compiling this positive utility on a formula into a utility on a single fluent as described above. This makes use of the fact that the negation of a formula in CNF is a formula in DNF and vice versa.

7.5 Related Work

The problem of planning with soft goals is also known as "planning with goal preferences", "partial satisfaction planning", and "oversubscription planning", and has previously been studied in several works. One popular approach has been to heuristically select a subset of the set of soft goals in the problem and treat them as hard goals to be planned towards by a standard classical planner. van den Briel et al. (2004) propose constructing this goal set incrementally, maintaining at each step a relaxed plan and the list of soft goals it achieves, and adding a soft goal to the set when the estimated utility of the plan can be increased. Later work extends their goal selection algorithm to incrementally consider sets of goals and take into account negative interactions or mutexes between them (Sanchez and Kambhampati, 2005). Smith (2004) select the set of goals by first converting the problem into an *orienteering* problem and solving it to obtain both a list of soft goals to be achieved and an order in which to achieve them. The goals are then passed to the planner one after the other in the suggested order.

The methods discussed above typically use heuristic search methods to solve the obtained problem that has only hard goals. An alternative to this is to integrate information about the soft goals directly into the heuristic, and avoid committing beforehand to a subset of soft goals. In this vein, van den Briel et al. (2004) propose a heuristic based on the relaxed planning graph which first constructs a relaxed plan π for all of the soft goals of the problem, and then performs a second pass to check for each soft goal p_i whether the cost of the actions in π that contribute only to achieving p_i exceeds the utility associated with p_i. Bonet and Geffner (2008) obtain a propositional encoding of the delete relaxation problem that can include costs and penalties on the fluents of the problem. This is then compiled in exponential time into a form which allows the (admissible) optimal delete-relaxation heuristic to be extracted from the model in linear time at each state.

An alternative compilation approach that has been proposed is to associate with each soft goal p_i a numeric variable n_i with domain $\{0, 1\}$ (Edelkamp, 2006). The utility for a plan is then expressed as $u(\pi) = \sum_{i=1}^{n} n_i * u(p_i) - cost(\pi)$, where $u(p_i)$ represents the utility associated with soft goal p_i and n_i represents the value of the numeric variable in the final state achieved by the plan. This transformation eliminates soft goals at the cost of introducing a plan metric that contains state-dependent terms (the values of the variables n_i). Current planners, however, are unable to effectively optimize for such metrics.

7.6 Conclusions

We have shown that soft goals, previously thought to add expressive power to the classical planning formalism, can be compiled away to yield a classical planning problem with action costs whose solutions encode solutions to the original problem, with only a linear increase in problem size. This planning problem can then be solved by off-the-shelf planners, which our results indicate perform much better on these problems than planning algorithms that are specifically designed to consider soft goals. A simple extension to our method also accommodates positive or negative utilities, in other words penalties, on CNF or DNF formulas made true by plans.

Part V

Conclusions and Future Work

Chapter 8

Conclusions

We now summarize the contributions of the thesis, and briefly discuss some ways in which the work presented here may be extended.

8.1 Contributions

The contributions of this thesis that we feel to be of greatest interest to the planning community are the following:

1. The computation of *relaxed plans from the additive and max heuristics* h^{add} *and* $h^{\max}$, which unifies the previously existing h^{add} and h^{ff} heuristics, and offers the advantages of both, namely cost-sensitivity, declarative, rather than algorithmic definition, explicit relaxed plans, and hence, no overcounting (in Chapter 3, and Keyder and Geffner (2008)).

2. The *set-additive heuristic* h^{sa}, which moves away from the independence assumption for costs and instead applies it to relaxed plans, obtaining closer approximations of the optimal cost for the delete relaxation (in Chapter 3, and Keyder and Geffner (2007, 2008))

3. The characterization of previously existing delete-relaxation heuristics in terms of a best supporter function that is dependent upon the independence assumption and the relaxed plan extraction algorithm, and the relation of the independence assumption to concepts from research into hypergraph problems (in Chapter 3 and Keyder and Geffner (2008))).

4. The *Steiner tree improvement procedure*, the first algorithm to allow the computation of suboptimal delete relaxation plans that do not make use of the independence assumption (in Chapter 4, and Keyder and Geffner (2009b)).

5. The declarative definition of the *complete set of causal delete relaxation landmarks*, and the use of this definition to compute this set with algorithms similar to those used for heuristic computation, without reliance on the relaxed planning graph (in Chapter 5, and Keyder et al. (2010)).

6. The use of our methods for finding delete relaxation landmarks to compute landmarks for the Π^m problem, which constitutes the first method for finding both *conjunctive landmarks* and *landmarks beyond the delete relaxation* (in Chapter 5, and Keyder et al. (2010)).

7. The characterization of a previously unexplored type of invariant in planning that we call *choice variables*, and a heuristic that exploits this invariant to obtain more accurate estimates. Furthermore, our work allows problems such as constraint satisfaction and graphical models to be embedded in planning problems and solved efficiently, facilitating the use of planning methods by researchers in other fields (in Chapter 6).

8. A compilation that shows that *soft goals do not add expressive power to the planning problem* and can be compiled away to obtain classical planning problems with action costs that can be solved with off-the-shelf planners, which perform better in the compiled problems than algorithms specifically designed for dealing with soft goals (in Chapter 7, and Keyder and Geffner (2009a)).

8.2 Future Work

The work described in this thesis suggests a number of extensions and directions for future work. Here we review some possibilities. The order of presentation follows the order in which the issues discussed appeared in the dissertation.

Exploiting the Independence Assumption

In Chapter 3, we introduced the set-additive heuristic that propagates sets of actions constituting relaxed plans for the fluents of the problem to obtain a global relaxed plan. We have also investigated the propagation of other types of information in these sets. Specifically, by choosing a pairwise mutex set of fluents in the problem that make up a multivalued variable and propagating these fluents themselves rather than the operators that achieve them when they are encountered as preconditions of an operator, heuristic estimates that respect delete information concerning these fluents can be obtained (Keyder and Geffner, 2007). While the optimal delete relaxation plan for achieving the required values of such a variable is a minimum spanning tree over the graph defined by its transitions, the optimal non-relaxed plan constitutes a *tour*, or a solution to the travelling salesman problem (TSP) (Cormen et al., 2001). While the TSP is intractable, several polynomial heuristics that give good results in practice have been proposed (Lin and Kernighan, 1973). A suboptimal tour over the values contained in a set can then be computed whenever a new plan is computed as the union of the relaxed plans of the preconditions of an operator, and its cost summed with the cost of the operators in the set to obtain the cost of the semi-relaxed plan.

In our original formulation, when choosing multivalued variables over whose values to compute a tour, we considered only those multivalued variables that form the leaves of the causal graph, in other words whose values can be changed with operators that have no preconditions upon other variables. Vehicles in transport problems, for example, fall into this category. How to compute the values of this heuristic in a more general setting, for example when the causal graph is cyclic, is an open question. A second issue is the automatic selection of variables that provide significant benefit when delete effects are considered, yet do not result in TSPs that are too large to solve approximately in practice.

Beyond the Independence Assumption

Our improvement procedure based on Steiner trees (Chapter 4) is the first method proposed until now for obtaining estimates for the delete relaxation of a problem that goes beyond the independence assumption without paying the exponential overhead of guaranteeing optimality. Several modifications to the procedure may be considered in order to further improve the quality of the estimation or lower its computational cost. Addressing first the issue of quality, recall that the Steiner tree procedure partitions the relaxed plan π into three sets in terms of a fluent y that is the precondition of some operator $o \in \pi$: $\pi^-(y)$, the portion of the plan required only in order to achieve y, $\pi^+(y)$, the portion of the plan that depends on y, and $\pi^0(y)$, the portion of the plan that falls into neither of the two categories. We considered an improvement procedure based on the idea of improving $\pi^-(y)$, yet there is no reason not to consider improvements to any one of the plan portions. One approach would be to remove both $\pi^-(y)$ and $\pi^+(y)$ from the plan and to attempt to find a lower cost plan for the goals that were previously achieved by $\pi^+(y)$ in the context of the state s augmented with the fluents supported by $\pi^0(y)$. As each improvement step guarantees a decrease in global plan cost, calls to each type of improvement algorithm could be interleaved until a fixpoint in which no improvement is possible is reached.

A second challenge in this area is to obtain better relaxed plans that do not necessarily contain the fluents present in a best plan according to some heuristic based on the independence assumption. For an example of this, consider Figure 8.1. As no plan obtained by a heuristic based on the independence assumption contains an operator with precondition p, the Steiner tree improvement procedure cannot obtain the optimal plan for this problem. This can be generalized to delete relaxation problems

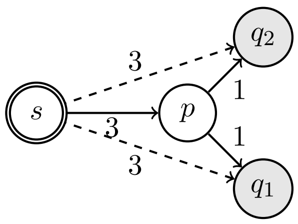


Figure 8.1: Goals of the problem are $\{q_1, q_2\}$. The Steiner tree improvement procedure is unable to obtain this plan as the fluent p required for the optimal delete relaxation plan is not present in any of the plans based on the independence assumption.

in which the optimal plan does not overlap with the best plans for the individual goals or preconditions.

Finally, the coverage obtained with this heuristic could be improved by lowering the number of improvements made to the relaxed plan, perhaps at the cost of worse quality. For this, the fluent y could be chosen from a subset of the preconditions of operators in the plan, rather than all possible preconditions. In this context, it makes sense to consider only those fluents y which are guaranteed to be in any valid relaxed plan, for example the goals of the problem. A second option is to consider only its delete-relaxation landmarks (Chapter 5).

Landmarks for the Delete Relaxation and Beyond

In Chapter 5, we discuss how to obtain landmarks for the global planning problem that are not necessarily landmarks for its delete relaxation. The approach suffers from two drawbacks. The first of these is that these landmarks, while resulting in a significantly more informative admissible heuristic when used in combination with optimal cost partitioning, do not perform well when used with the uniform cost partitioning, which is much cheaper to compute and results in better coverage. A possible avenue of further research is then to investigate new cost partitioning schemes which result in more informative heuristic estimates than the uniform one without paying the computational overhead of guaranteeing optimality.

A second challenge for the Π^m landmarks is that the delete relaxation problem defined by Π^m for $m > 1$ is much larger than than the delete relaxation problem Π^+. The propagation procedure used to find landmarks is then impractical, as it does not reach a fixpoint within a reasonable amount of time. One possible way of addressing this issue is to use a Π^m problem that preserves the completeness and soundness of the landmarks that are found, yet does not contain fluents and operators that are redundant in the context of landmark finding. For an example of this, consider a logistics problem with two trucks t_1 and t_2, and the Π^m fluents made up of the simultaneous position of the two trucks, i.e. fluents of the form $f^m = (\mathit{att}_1\mathit{loc}_i, \mathit{att}_2\mathit{loc}_j)$. It can be shown that the set of landmarks for such a fluent always consists of the union of the landmarks for the two fluents considered separately, that is, $\mathit{LM}(f^m) = \mathit{LM}(\mathit{att}_1\mathit{loc}_i) \cup \mathit{LM}(\mathit{att}_2\mathit{loc}_j)$. The removal of such fluents then results in a smaller delete relaxation problem with no loss of completeness or soundness. A more general issue is to define *independence* criteria between fluents for different types of computations, such as the use of the Π^m graph to obtain the h^m heuristic in which the cost of a set is equal to the most costly set of m fluents in the set.

Finally, here we have investigated the use of Π^m landmarks only in the context of generating admissible heuristics. Effective ways of using them in the satisficing planning setting therefore remains an open problem.

Heuristics with Choice Variables

In our presentation of choice variables (Chapter 6), we assumed that they were given to the planner as additional information along with the problem representation. To make our formulation truly domain-independent, we plan to investigate in the future methods for detecting this type of structure in the problem and choosing the choice variables to be used accordingly. One challenge in this area will be to select choice variables while imposing a limit upon the width of the resulting choice variables graph G_C.

We have also considered the idea of choice variables that can be assigned a value more than once in a valid plan. This would allow the method to be applied to a greater range of planning problems with the puzzle-like structure in which the type of reasoning performed by the choice variables heuristic is effective. A second way of extending the scope of this heuristic is to consider as choice variable sets the possible *achievers* of a fluent for a given instance in which it must be achieved. One example of this would be to treat as a choice variable the possible achievers of a goal or landmark fluent, and reason about the cost of the rest of the plan given a commitment to achieving it with a certain operator.

8.3 Open Challenges

While the heuristic search approach to classical planning has led to unprecedented increases in the size and range of problems that can be solved with planning techniques, in recent years the benefits that can be obtained from ever more accurate admissible or non-admissible approximations to the optimal cost of the delete relaxation have diminished. Substantial improvements in performance have then come from improvements to search such as the introduction of enforced hill climbing (Hoffmann and Nebel, 2001), delayed evaluation and multiple open lists (Richter and Westphal, 2010), and approaches that take into account problem structure in the form of landmarks (Richter and Westphal, 2010; Helmert and Domshlak, 2009) as well as causal chains and delete information such as mutexes (Lipovetzky and Geffner, 2009). The work presented here on landmarks that take into account delete information (Chapter 5) and other new invariants in planning problems (Chapter 6), as well as some of the work that not included in this thesis that we discuss above (Section 8.2) also represents steps in this direction.

However, important problems remain to be solved in this area. Principal among these is the application of inference to puzzle-like problems, which can be broadly characterized as the class of problems in which achieving one subgoal requires undoing work that has been done before to achieve other goals, which must then be reachieved. While domain-specific heuristic search techniques have had great success in problems that fit this criteria such as the 15-puzzle and the Rubik's cube problems, domain-independent problem solving techniques still have a long way to go to match their performance. Existing inference techniques that take into account delete information are either impractical, as they are too computationally expensive to be applied to large problems, or require specific domain features that are not generally present.

One approach to the first problem that we believe will yield substantial returns is to develop methods for identifying the components of the problem that exhibit this type of structure, and solving *only these components* with inference methods that are able to meaningfully reason about their interactions. The remaining parts of the problem can then be solved with more scalable delete relaxation techniques. Both the choice variables heuristic presented in Chapter 6 and the TSP heuristic presented above are examples of this type of decomposition; the challenge here is to identify automatically the parts of the problem that merit further computational effort and integrate the solutions to these into solutions for the rest of the problem.

Another challenge here is to automatically detect the *type* of inference that is required for the component: whether inference concerning the changes in value of a single variable, such as in the TSP heuristic, is sufficient, whether interactions among graphical model-type variables, such as in the choice variables heuristic, must be considered, or whether more general inference about delete effects, such as that performed by the h^m family of heuristics, is required.

We look forward to facing these challenges.

Part VI

Appendix

Bibliography

Eyal Amir and Barbara Engelhardt. Factored planning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 929–935. Morgan Kaufmann, 2003. 104, 130

Giorgio Ausiello, Giuseppe F. Italiano, and Umberto Nanni. Hypergraph traversal revisited: Cost measures and dynamic algorithms. In *Lecture Notes in Computer Science*, pages 1–16, 1998. 19, 21, 29, 55

Christer Bäckström and Bernhard Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995. 6

Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958. 37

J. Benton, Minh Do, and Subbarao Kambhampati. Anytime heuristic search for partial satisfaction planning. *Artificial Intelligence*, 173(5-6):562–592, April 2009. 145, 146

Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence – IJCAI-95*, pages 1636–1642. Morgan Kaufmann, 1995. 53

Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–23, 1993. 115

Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001. 8, 17, 18, 21, 31, 34, 53

Blai Bonet and Hector Geffner. Heuristics for planning with penalties and rewards formulated in logic and computed through circuits. *Artificial Intelligence*, 172(12-13):1579–1604,

2008. 148

Blai Bonet and Malte Helmert. Strengthening landmark heuristics via hitting sets. In *Proceedings of the Twentieth European Conference on Artificial Intelligence (ECAI 2010)*, pages 329–334, 2010. 99

Blai Bonet, Gabor Loerincs, and Hector Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI 1997)*, pages 714–719, 1997. ix

Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in bayesian networks. In *Proceedings of the Twelfth Conference in Uncertainty in Artificial Intelligence (UAI 1996)*, pages 115–123, 1996. 104

Ronen I. Brafman and Carmel Domshlak. Factored planning: How, when, and when not. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006)*, pages 809–814, 2006. 104, 131

Tom Bylander. The computational complexity of STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994. 6, 7, 18, 19

Moses Charikar, Chandra Chekuri, Toyat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed Steiner problems. *Journal of Algorithms*, pages 73–91, 1998. 25, 59

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2001. 6, 23, 30, 37, 57, 58, 59, 155

Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001. 104, 113, 114, 118

Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 37

Minh B. Do and Subbarao Kambhampati. Sapa: A scalable multi-objective heuristic metric temporal planner. *Journal of Artificial Intelligence Research*, 20:2003, 2003. 54

Stefan Edelkamp. On the compilation of plan constraints and preferences. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pages 374–377. AAAI Press, 2006. 148

Stefan Edelkamp and Shahid Jabbar. MIPS-XXL: Featuring external shortest path search for sequential optimal plans and external branch-and-bound for optimal net benefit. In *Sixth International Planning Competition Booklet (ICAPS 2008)*, 2008. 143

Stefan Edelkamp and Peter Kissmann. Gamer: Bridging planning and general game playing with symbolic search. In *Sixth International Planning Competition Booklet (ICAPS 2008)*, 2008. 143

Eric Fabre, Loig Jezequel, Patrik Haslum, and Sylvie Thiébaux. Cost-optimal factored planning: Promises and pitfalls. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pages 65–72, 2010. 104

Yousri El Fattah and Rina Dechter. An evaluation of structural parameters for probabilistic reasoning: Results on benchmark circuits. In *Proceedings of the Twelfth Conference in Uncertainty in Artificial Intelligence (UAI 1996)*, pages 244–251, 1996. 116

Richard Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971. 5

Raquel Fuentetaja, Daniel Borrajo, and Carlos Linares L'opez. A new approach to heuristic estimations for cost-based planning. In *Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference*, pages 543–548, 2008. 55

Giorgio Gallo, Giustino Longo, Sang Nguyen, and Stefano Pallottino. Directed hypergraphs and applications, 1992. 19

Cenk Gazen and Craig Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *Proceedings of the Fourth European Conference on Planning (ECP 1997)*, pages 221–233, 1997. 137, 147

Alfonso Gerevini and Derek Long. Preferences and soft constraints in PDDL3. In *Proceedings of the ICAPS-06 Workshop on Preferences and Soft Constraints in Planning*, pages 46–53, 2006. 143

Patrik Haslum. Additive and reversed relaxed reachability heuristics revisited. In *Sixth International Planning Competi-*

tion Booklet (ICAPS 2008), 2008. 143

Patrik Haslum. $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 354–357, 2009. 87

Patrik Haslum and Héctor Geffner. Admissible heuristics for optimal planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*, pages 140–149, 2000. 86, 102

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006. 8, 12, 13, 102

Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 2009. ix, 98, 102, 159

Malte Helmert, Minh Do, and Ioannis Refanidis. IPC 2008 deterministic competition. In *Sixth International Planning Competition Booklet (ICAPS 2008)*, 2008. 143, 144

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001. 7, 12, 17, 34, 40, 54, 159

Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22(1):215–278, 2004. 80, 81, 95, 97, 101

Richard Howey and Derek Long. VAL's progress: The automatic validation tool for PDDL2.1 used in the international planning competition. In *Proceedings of the ICAPS 2003 Workshop on the Competition: Impact, Organisation, Evaluation, Benchmarks*, 2003. 144

Chih-Wei Hsu and Benjamin W. Wah. The SGPlan planning system in IPC6. In *Sixth International Planning Competition Booklet (ICAPS 2008)*, 2008. 145

Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. 58

Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1728–1733, 2009. 94, 98, 101

Emil Keyder and Hector Geffner. Set-additive and TSP heuristics for planning with action costs and soft goals. In *ICAPS 2007 Workshop on Heuristics for Domain-Independent Planning: Progress, Ideas, Limitations, Challenges*, 2007. 146, 153, 155

Emil Keyder and Héctor Geffner. Heuristics for planning with action costs revisited. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pages 588–592, 2008. 153, 154

Emil Keyder and Hector Geffner. Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, 36:547–556, 2009a. 154

Emil Keyder and Hector Geffner. Trees of shortest paths vs. steiner trees: Understanding and improving delete relaxation heuristics. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1734–1739, 2009b. 154

Emil Keyder, Silvia Richter, and Malte Helmert. Sound and complete landmarks for and/or graphs. In *Proceedings of the Twentieth European Conference on Artificial Intelligence (ECAI 2010)*, pages 335–340, 2010. 154

Donald E. Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5, 1977. 37, 38

Xiao Yu Li. *Optimization Algorithms for the Minimum-Cost Satisfiability Problem.* PhD thesis, North Carolina State University, 2004. 121

Shen Lin and Brian Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21: 498–516, 1973. 155

Nir Lipovetzky and Hector Geffner. Inference and decomposition in planning using causal consistent chains. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 2009. 102, 159

Yaxin Liu, Sven Koenig, and David Furcy. Speeding up the calculation of heuristics for heuristic search-based planning. In

Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI 2002), pages 484–491. AAAI Press, 2002. 37, 72, 85

Drew McDermott. A heuristic estimator for means-ends analysis in planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, 1996. ix

David G. Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions of sat problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI 1992)*, pages 459–465, 1992. 126

Bernard Nebel. Compilation schemes: A theoretical tool for assessing the expressive power of planning formalisms. In *Proceedings KI-99: Advances in Artificial Intelligence*, pages 183–194. Springer-Verlag, 1999. 147

Bernhard Nebel. On the compilability and expressive power of propositional planning. *Journal of Artificial Intelligence Research*, 12:271–315, 2000. 137

Judea Pearl. *Heuristics*. Addison Wesley, 1983. 8, 9, 10, 11, 17

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-73-7. 104

Julie Porteous, Laura Sebastia, and Jörg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In Amedeo Cesta and Daniel Borrajo, editors, *Pre-proceedings of the Sixth European Conference on Planning (ECP 2001)*, pages 37–48, 2001. 95

H. Proemel and A. Steger. *The Steiner Tree Problem: A Tour Through Graphs, Algorithms, and Complexity*. Vieweg+Teubner Verlag, 2002. 25, 58, 59

Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010. ix, 7, 13, 17, 45, 54, 101, 159

Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 975–

982, 2008. 82, 89, 95, 98

Gabriel Robins and Alexander Zelikovsky. Improved Steiner tree approximation in graphs. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000)*, pages 770–779, 2000. 25, 59

Romeo Sanchez and Subbarao Kambhampati. Planning graph heuristics for selecting objectives in over-subscription planning problems. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, pages 192–201, 2005. 148

David E. Smith. Choosing objectives in over-subscription planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 393–401, 2004. 148

Menkes van den Briel, Romeo Sanchez, Minh B. Do, and Subbarao Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, pages 562–569, 2004. 148

Vincent Vidal and Hector Geffner. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence*, 170 (3):289–335, 2006. 102

Alexander Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1): 99–110, 1997. 25

Lin Zhu and Robert Givan. Landmark extraction via planning graph propagation. In *ICAPS 2003 Doctoral Consortium*, pages 156–160, 2003. 83, 95, 98