Non-classical Heuristics for Classical Planning

Erez Karpas

Non-classical Heuristics for Classical Planning

Research Thesis

In Partial Fulfillment of the
Requirements for the
Degree of Doctor of Philosophy

Erez Karpas

Submitted to the Senate of
the Technion - Israel Institute of Technology

Adar, 5772 Haifa March 2012

The Research Thesis Was Done Under The Supervision of Prof. Carmel Domshlak and Prof. Shaul Markovitch in the Faculty of Industrial Engineering and Management.

First and foremost, I would like to thank my advisors, Carmel Domshlak and Shaul Markovitch. Without their guidance, this thesis would not exist, and I would be a different person. I would also like to thank the other members of my thesis committee, Malte Helmert and Moshe Tennenholtz, for providing invaluable advice.

I have collaborated with many peple during this time, and have learned something from each of them — they all deserve my thanks: Roei Bahumi, Ziv Even-Zur, Chris Fawcett, Yannai Golany, Malte Helmert, Holger Hoos, Michael Katz, Emil Keyder, Yevegni Nus, Silvia Richter, Gabriele Röger, and Jendrik Seipp. I have also had many an interesting discussion (and a few beers) during the ICAPS conferences. I am honored to belong to such a great community, and look forward to many more meetings.

Last, but not least, Dikla and my family deserve special thanks — I could not have not this without you.

The Generous Financial Help Of Technion Is Gratefully Acknowledged.

PUBLICATIONS

Refereed Papers in Journals

1. E. Karpas, S.E. Shimony and A. Beimel, *Approximate belief updating in max-2-connected Bayes networks is NP-hard*, in Artificial Intelligence 173 (2009) 1150-1153

Refereed Papers in Conference Proceedings

1. E. Karpas, M. Katz and S. Markovitch, *When Optimal is Just Not Good Enough: Fast Near-Optimal Action Cost-Partitioning*, in ICAPS 2011, 21st International Conference on Automated Planning and Scheduling, Freiburg, Germany, June 2011

2. E. Karpas, C. Domshlak and S. Markovitch, *To Max or not to Max: Online Learning for Speeding Up Optimal Planning*, in AAAI 2010, 24th AAAI Conference on Artificial Intelligence, Atlanta, Georgia, USA, July 2010

3. E. Karpas and C. Domshlak, *Cost-optimal Planning with Landmarks*, in IJCAI 2009, 21st International Joint Conference on Artificial Intelligence, Pasadena, CA, USA, July 2009

Papers in Conference Workshop Proceedings

1. E. Karpas and C. Domshlak, *Optimal Planning with Inadmissible Heuristics*, ICAPS 2012, 22nd International Conference on Automated Planning and Scheduling, Atibaia, Sao Paulo, Brazil, June 2012 (to appear)

2. E. Karpas and C. Domshlak, *Living on the Edge: Safe Search with Unsafe Heuristics*, in Workshop on Heuristics for Domain Independent Planning, ICAPS 2011, 21st International Conference on Automated Planning and Scheduling, Freiburg, Germany, June 2011

3. C. Domshlak, Z. Even-Zur, Y. Golany, E. Karpas and Y. Nus, *Command and Control Training Centers: Computer Generated Forces Meet Classical Planning*, in Workshop on Scheduling and Planning Applications, ICAPS 2011, 21st International Conference on Automated Planning and Scheduling, Freiburg, Germany, June 2011

4. C. Fawcett, M. Helmert, H. Hoos, E. Karpas, G. Röger and J. Seipp, *FD-Autotune: Domain-Specific Configuration using Fast Downward*, in Workshop on Learning and Planning, ICAPS 2011, 21st International Conference on Automated Planning and Scheduling, Freiburg, Germany, June 2011

5. M. Helmert, G. Röger and E. Karpas, *Fast Downward Stone Soup: A Baseline for Building Planner Portfolios*, in Workshop on Learning and Planning, ICAPS 2011, 21st International Conference on Automated Planning and Scheduling, Freiburg, Germany, June 2011

6. E. Karpas, C. Domshlak and S. Markovitch, *Learning to Combine Admissible Heuristics Under Bounded Time*, in Workshop on Planning and Learning, ICAPS 2009, 19th International Conference on Automated Planning and Scheduling, Thessaloniki, Greece, September 2009

Working Papers

1. R. Bahumi, C. Domshlak and E. Karpas *Deeply Preferred Operators: Lazy Search Meets Lookahead*, submitted to ICAPS 2012 Workshop on Heuristics and Search for Domain-Independent Planning

2. C. Domshlak, S. Markovitch and E. Karpas *Online Speedup Learning for Optimal Planning*, Under preparation

3. C. Domshlak, M. Helmert, E. Karpas, E. Keyder and S. Richter *Landmarks in Optimal Planning*, Under preparation

Contents

List of Figures

List of Tables

Abstract

Domain-independent planning is one of the foundational areas in the field of Artificial Intelligence (AI). A planning task consists of an initial world state, a goal, and a set of actions for modifying the world state, with the objective of finding a plan that transforms the initial world state into a goal state. In cost-optimal planning, we are interested in finding not just any valid plan, but a cheapest such plan. One of the most prominent approaches to cost-optimal planning these days is heuristic state-space search, guided by a heuristic which estimates the distance from each state to the goal. Most heuristics for domain-independent planning are what we call classical — they estimate the distance from some given state to the goal using only properties of the given state. In this work, we explore non-classical heuristics — heuristics which exploit additional information gathered during search. We propose a mathematical model which allows us to formally define non-classical heuristics, as well as a useful taxonomy of heuristics along several dimensions. We then describe two different classes of non-classical heuristics: landmark-based heuristics, and machine-learning based heuristics. Our empirical evaluation shows that non-classical heuristics are not just an interesting theoretical possibility, but rather state of the art tools in heuristic search planning.

Chapter 1

Introduction

Automated domain-independent planning is the model based approach to autonomous behavior (Geffner, 2010), and is one of the most important and well-studied problems in the field of artificial intelligence. Automated planning deals with achieving a certain goal from a specified initial state, using a set of given operators. Several different types of models exist, with the most complex featuring partial observability of the world states, non-deterministic transitions, temporal processes, and more. Classical planning is perhaps the simplest possible model — featuring full observability, fully deterministic actions, and sequential actions. Yet even in this simple model, determining whether a given planning task has a solution is computationally hard (Bylander, 1994).

In cost-optimal planning, the objective is not just to find any plan, but rather to find one of the best (that is, cheapest) possible plans. On the other hand, in satisficing planning, sub-optimal plans are valid solutions. In general, both satisficing and cost-optimal planning are equally computationally difficult. However, in many planning benchmarks, cost-optimal planning is computationally hard, while satisficing planning is tractable (Helmert, 2008).

Our focus is on cost-optimal planning. In recent years, great progress in cost-optimal planning has been made, due to new methods for automatic derivation of admissible heuristics for domain-independent planning: h^{max} (Bonet and Geffner, 2001), h^m (Haslum and Geffner, 2000), Pattern databases (Edelkamp, 2001), Merge and shrink abstractions (Helmert et al., 2007), Implicit abstractions (Katz and Domshlak, 2010b), and $h_{\text{LM-CUT}}$ (Helmert and Domshlak, 2009). The above heuristics, although being a partial list, cover a wide range of heuristics, representing different families

of heuristics. Nevertheless, they all share a common attribute — they are all what we call *classical* heuristics, which estimate the distance from some given state to the goal using only properties of the given state.

In this work, we explore *non-classical* heuristics — heuristics which exploit additional information gathered during search. To our surprise, the heuristic search literature does not provide a formal mathematical definition which allows us to formally describe non-classical heuristics. Therefore, after a brief presentation of the necessary background in Chapter 2, our first contribution, in Chapter 3, is such a formal model, which allows us to define non-classical heuristics and their properties with mathematical notation. We also discuss a taxonomy of heuristic evaluation functions along several dimensions, and present new search algorithms, which are better suited for use with non-classical heuristics.

In order to show that non-classical heuristics are not just theoretically possible, but also a natural way to exploit information gathered during search, we present two different classes of non-classical heuristics. In Chapter 4 we present landmark based heuristics, which exploit information gathered from the known path (or paths) to the state being evaluated. In Chapter 5 we present machine-learning based heuristics, which exploit information gathered throughout the search, not just from paths leading to the state being evaluated. In Chapter 6 we present an empirical evaluation, which demonstrates that these two classes of heuristics are state of the art. Finally, we present our conclusions in Chapter 7.

Chapter 2

Background

In this chapter we will formally define the problem we attempt to solve — domain independent planning, as well as give background on search problems.

2.1 Domain Independent Planning

A domain independent planning task (*planning task* or simply *task* for short) consists of a description of an initial state, a goal, and a set of available operators. In this work, we will use the STRIPS formalism (Fikes and Nilsson, 1971) extended with action costs. Most of this work applies to other well known formalisms, such as ADL (Pednault, 1989) and SAS$^+$ (Bäckström and Klein, 1991; Bäckström and Nebel, 1995) with minor modifications. Our notations mostly follow Helmert and Domshlak (2009).

A STRIPS with action costs planning task is a 5-tuple $\Pi = \langle P, I, G, A, \mathcal{C} \rangle$, where P is a set of propositions (boolean atoms). A state $s \subseteq P$ is a subset of propositions, with the interpretation that the propositions in s are true and all other propositions are false. I is a state, called the *initial state*, and $G \subseteq P$ is called the *goal*. For ease of notation and without loss of generality, we assume that there is a single goal proposition ($G = \{p_g\}$), which can only be achieved by a single action, END.

A is a set of actions, each of which is a triple $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$, with $\mathsf{pre}(a) \subseteq P$, $\mathsf{add}(a) \subseteq P$, and $\mathsf{del}(a) \subseteq P$. An action a is applicable in state s if $\mathsf{pre}(a) \subseteq s$. If action a is applied in state s, it results in the new state $s' = (s \setminus \mathsf{del}(a)) \cup \mathsf{add}(a)$. $\mathcal{C} : A \to \mathbb{R}^{0+}$ is a cost function on actions, which assigns each action a non-negative cost.

A sequence of actions $\pi = \langle a_0, a_1, \ldots, a_n \rangle$ is an s-path if a_0 is applicable

in state s and results in state s_1, a_1 is applicable in s_1 and results in s_2, and so on. The state resulting from applying action sequence π in state s is denoted by $s[\![\pi]\!]$. If π_1 and π_2 are action sequences, by $\pi_1 \cdot \pi_2$ we denote the concatenation of π_1 and π_2. The cost of action sequence $\langle a_0, a_1, \ldots, a_n \rangle$ is $\sum_{i=0}^{n} \mathcal{C}(a_i)$. An s-path π is an s-plan for Π if $G \subseteq s[\![\pi]\!]$. An optimal plan is a cheapest such I-plan.

A useful notion in domain-independent planning is that of a *causal link*. Let $\pi = \langle a_0, a_1, \ldots a_n \rangle$ be an s-path. The triple $\langle a_i, p, a_j \rangle$ forms a causal link in π if $i < j$, $p \in \mathsf{add}(a_i)$, $p \in \mathsf{pre}(a_j)$, $p \notin s[\![\langle a_0, a_1, \ldots a_{i-1} \rangle]\!]$, and for $i < k < j$, $p \notin \mathsf{del}(a_k) \cup \mathsf{add}(a_k)$. In other words, a_i is the actual provider of precondition p for a_j. In such a causal link, a_i is called the *provider*, and a_j is called the *consumer*. In Section 4.4, we will show one possible use for causal links.

There are several approaches to solving domain independent planning tasks. The most prominent approach, with which this work is concerned, is heuristic forward search. Before we describe this approach in detail, it is important to mention that other methods of solving planning tasks exist. For example, it is possible to compile a planning task into an instance (or series of instances) of another problem, such as boolean satisfiability testing (SAT) (Kautz and Selman, 1992) or constraint programming (CP) (Vidal and Geffner, 2006). Other approaches exploit symbolic search (Edelkamp and Helmert, 2001) and/or bidirectional search (Kissmann and Edelkamp, 2011). However, as evidenced by both the results and the participating planners in the last several international planning competitions (IPC), heuristic forward search is the state of the art in cost-optimal planning.

2.2 Search Problems

A planning task can be easily seen as a search problem. We will first formally describe search problems, and then show how one can view a planning task as a search problem. A search problem is a 6-tuple $\mathbf{P} = \langle S, I, G, A, f, \mathcal{C} \rangle$, where S is a set of *states* (also known as the *state space* or *search space*), $I \in S$ is called the *initial state*, and $G \subseteq S$ is a set of *goal states*. A is a set of actions and $f : S \times A \to S$ is the *transition function*, such that applying action a in state s leads to state $f(s, a)$. $\mathcal{C} : A \to \mathbb{R}^{0+}$ is a cost function on actions, which assigns each action a non-negative cost.

The outgoing transitions from s, denoted by $\mathsf{succ}^{\mathsf{a}}(s)$, are defined as $\mathsf{succ}^{\mathsf{a}}(s) := \{\langle a, s' \rangle \mid a \in A, f(s, a) = s'\}$, and the successors of s, denoted $\mathsf{succ}(s)$, are simply the states for which there is an outgoing transition

$\mathsf{succ}(s) := \{s' \mid \exists a : \langle a, s' \rangle \in \mathsf{succ}^{\mathsf{a}}(s)\}$.

Let s be some state. A sequence of actions $\langle a_0, \ldots, a_n \rangle$ is an s-path if $f(\ldots f(f(s, a_0), a_1), \ldots, a_n)$ is well defined. We denote by Γ the set of all valid I-paths. A sequence of actions $\langle a_0, \ldots, a_n \rangle$ is an s-plan if $f(\ldots f(f(I, a_0), a_1), \ldots, a_n) \in G$. An I-plan is a solution for our search problem. The cost of such a sequence of actions is $\sum_{i=0}^{n} \mathcal{C}(a_i)$, and a solution is said to be optimal if no lower-cost solution exists.

Given a planning task $\Pi = \langle P, I, G, A, \mathcal{C} \rangle$ it is very easy to see how to create a corresponding search problem $\mathbf{P} = \langle S, I', G', A', f, \mathcal{C}' \rangle$. The states of the search problem are simply subsets of the propositions of the planning task, so $S = 2^P$. The initial state of the search problem is the initial state of the planning task, so $I' = I$. The goal states of the search problem are those in which the goal of the planning task holds, so $G' = \{s \in S | G \subseteq s\}$. The actions of the search problem are the same as those of the planning task, so $A' = A$, and the transition function f is uniquely determined by the actions' preconditions, add and delete effects (for ease of presentation we will assume that inapplicable actions lead to the same state), so that

$$f(s, a) = \begin{cases} (s \setminus \mathsf{del}(a)) \cup \mathsf{add}(a) & \mathsf{pre}(a) \subseteq s \\ s & \text{otherwise,} \end{cases}$$

and the cost of an action in the search problem is the same as the cost of the corresponding action in the planning task, so $\mathcal{C}' = \mathcal{C}$. Clearly, any solution of the search problem is a solution for the planning task with the same cost, and so any optimal solution of the search problem is an optimal solution of the planning task.

Of course, if we want to solve a domain independent planning task as a search problem, we still need to know how to solve search problems. Fortunately, search is one of the fundamental topics of artificial intelligence, and it is fairly well understood. While it is possible to use uninformed search techniques, such as breadth first search or iterative deepening depth first search (Korf, 1985), the size of the state space grows exponentially with the number of propositions, and these methods quickly become infeasible.

Let us denote the cost of an optimal path going from state s to state s' by $k(s, s')$. Consider what we could do if we knew for each state s the true cheapest cost from s to the goal (denoted by $h^*(s) := \min_{s' \in G} k(s, s')$, which is also called the *perfect heuristic*). First, note that $c^* := h^*(I)$ is the cost of the optimal solution. At least one of the successors of I is on the optimal path to the closest goal state. In general, by going from each state s to a successor s' with the lowest value of $h^*(s')$, we would find an optimal

solution.

Unfortunately, it is usually not feasible to compute h^*. For example, STRIPS planning is PSPACE-complete (Bylander, 1994), implying that it is PSPACE-hard to compute the perfect heuristic. However, it is still possible to guide the search towards a solution more quickly using a *heuristic evaluation function* (or heuristic for short), which attempts to approximate h^*. A heuristic evaluation function associates with each state s in the search space an estimate of $h^*(s)$ — the distance from s to the nearest goal. Many well known search algorithms can exploit such a heuristic, such as greedy best first search and A^* (Hart et al., 1968). In the next chapter, we formally define heuristic evaluation functions, and discuss some well known search algorithms.

Chapter 3

Non-classical Heuristic Search

> "Heuristic search has been one of the important ideas to grow out of artificial intelligence research. It is an ill-defined concept..."
> (Pohl, 1970)

Generally speaking, heuristics are easy to compute methods which can help choose among alternate courses of action. However, in the context of heuristic search, heuristics are typically used to estimate the distance from search states to the goal. Many papers about heuristic search, and A^* in particular, either neglect to give a formal mathematical definition of heuristics, or state that heuristics are functions which map search states into non-negative numbers (Pohl, 1970; Harris, 1974; Martelli, 1977; Korf, 1985; Russell and Norvig, 2010; Edelkamp and Schrödl, 2011). Others note that heuristic estimates might also depend on some information gathered during the search, rather than only on the description of the evaluated state (Gelperin, 1977; Bagchi and Mahanti, 1983; Mérő, 1984; Dechter and Pearl, 1985). However, we are not familiar with any previous work which gives a formal mathematical definition of heuristics that takes into account the fact that heuristics can exploit information other than the state description[1]. In this chapter, we will provide such a formal mathematical definition of heuristic evaluation functions, discuss a new taxonomy of heuristic evalu-

[1]Some previous work refers to search *nodes*, which contain a description of the state as well as some additional information. However, what additional information is contained in the search node is not clear, and varies between different works. Even so, we are not aware of any definition of search node which allows what we will later call history dependent heuristics.

ation functions which results from our definition, and present appropriate search algorithms for dealing with these new types of heuristics. We remark that while the rest of this work is concerned specifically with domain-independent planning, this chapter is more general, and applies to heuristic search in general.

3.1 Mathematical Framework

Our mathematical formulation is based on the following informal verbal definition:

> "the promise of a node is estimated numerically by a **heuristic evaluation function** $f(n)$ which, in general, may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain." (Pearl, 1984)

Following the above definition, the list of which types of information heuristics might be based upon can be divided into three parts:

1. The description of the goal and any extra knowledge about the problem domain
2. The description of n
3. The information gathered by the search up to that point

For a fixed search problem, the first part can be seen as constant, since the goal and problem domain do not change. For domain independent planning in particular, this is simply the problem description (that is, the initial state, the goal, and descriptions of the actions), since it is assumed that there is no external knowledge about the problem except its description.

The second part (the description of the state) is what is usually considered to be the main source of information for a heuristic. Most heuristics for domain independent planning are based only upon the description of the state and of the problem (Bonet et al., 1997; Hoffmann and Nebel, 2001; Haslum and Geffner, 2000; Helmert, 2004; Helmert and Geffner, 2008; Helmert et al., 2007; Katz and Domshlak, 2010b).

The third part (the information gathered by the search up to that point) referring to the point in time when the heuristic estimate for a certain state was computed, is often overlooked. In fact, most heuristics described in the domain independent planning literature **do not** use any information

gathered by the search, and can be seen simply as functions mapping states to non-negative numbers. We refer to these heuristics as *classical heuristics*. In this work, we focus on heuristics which **do** use information gathered by the search — *non-classical heuristics*.

Before we can discuss classical vs. non-classical heuristics, we must first give a formal definition of heuristics, and specifically define the information gathered by the search at a certain point. We begin by defining the history of a search (which we will refer to as *search history*):

Definition 1 (Search History)
Let $\mathbf{P} = \langle S, I, G, A, f, \mathcal{C} \rangle$ *be a search problem. A sequence of states* $\omega = \langle s_0, s_1, \ldots s_n \rangle$ *is a possible history of* $\mathbf{P}$ *iff:*

1. $s_0 = I$
2. *For* $i = 1 \ldots n$, $\exists j < i$ *such that* $s_i \in \mathsf{succ}(s_j)$ *or* $s_i = s_j$

Definition 1 captures possible sequences of states which might be expanded by a forward search — every such sequence must begin with the initial state, and in order to expand some state, it must either have already been expanded (and is thus known to the search) or one of its parents must have already been expanded. We denote the set of all possible search histories of search problem $\mathbf{P}$ by $\mathcal{H}_{\mathbf{P}}$.

Based upon the notion of possible search history, we can now give a formal definition of heuristic evaluation functions. We take the view that a heuristic evaluation function evaluates a *path* rather than a state. This is the view taken by Dechter and Pearl (1985) for generalized best first search, and it allows us, for example, to define "the distance from the initial state" as a heuristic. Thus, a heuristic evaluation function is a function which maps a search history and a path to the set of non-negative real numbers. Formally:

Definition 2 (Heuristic Evaluation Function)
Let $\mathbf{P} = \langle S, I, G, A, f, \mathcal{C} \rangle$ *be a search problem. A* heuristic evaluation function *is a function* $h : \mathcal{H}_{\mathbf{P}} \times \Gamma \rightarrow \mathbb{R}^{0+}$.

In other words, a heuristic evaluates a path *given a search history*, and might produce two different evaluations for the same state given different paths to the same state, or even given the same path but with different histories. Using Definition 2, we can now return to our discussion of the information gathered during search. First, we define the sets of expanded and generated states, based upon the search history. Given a search history

$\omega = \langle s_0, s_1, \ldots s_n \rangle$, we denote by $Exp_\omega := \{s_0, \ldots, s_n\}$ the states expanded so far, and by $Gen_\omega := Exp_\omega \cup \bigcup_{s \in Exp_\omega} \mathsf{succ}(s)$ the states generated so far — the expanded states and their successors.

One type of information that could be gathered by forward search is the structure of the explored part of the search space. We denote the structure explored by search history ω by $\mathcal{G}_\omega$ — a directed graph, whose nodes consist of all generated states, and with edges consisting of all transitions outgoing from the expanded states. Formally $\mathcal{G}_\omega = \langle Gen_\omega, \{\langle s, s' \rangle \mid s \in Exp_\omega, s' \in \mathsf{succ}(s)\}\rangle$.

Additionally, there could be more information associated with each state. One common example of this is storing the heuristic value computed for previously encountered states. We formalize this by a function $\mathcal{I}_\omega$ which assigns some bit sequence for each state. Formally, $\mathcal{I}_\omega : Gen_\omega \to \mathcal{B}$, where $\mathcal{B}$ is the set of all possible bit sequences.

3.2 A Taxonomy of Heuristics

With a formal definition of heuristic evaluation functions at hand, we are ready to describe a taxonomy of heuristics along several dimensions. The first dimension we discuss is what type of information is used by the heuristic — what we call the *information dependence*.

3.2.1 Information Dependence

Recall that according to Definition 2, a heuristic takes two input parameters: a search history and a path. We first explore the dependency of a heuristic on the path being evaluated.

Definition 3 (Path Independent Heuristic)
Let $\mathbf{P} = \langle S, I, G, A, f, \mathcal{C} \rangle$ *be a search problem, and let* $h : \mathcal{H}_\mathbf{P} \times \Gamma \to \mathbb{R}^{0+}$ *be a heuristic evaluation function.* h *is called* path independent *iff for any two* I*-paths* $\pi_1, \pi_2 \in \Gamma$ *such that* $I[\![\pi_1]\!] = I[\![\pi_2]\!]$*, and for any search history* $\omega \in \mathcal{H}_\mathbf{P}$*,* $h(\omega, \pi_1) = h(\omega, \pi_2)$*.*

In other words, a heuristic is said to be path independent if the heuristic estimate does not depend on the path being evaluated, but only on the state the path reaches (and possibly on the search history). If the heuristic value does depend on the path being evaluated, we call h *path dependent*. We now turn our attention to the dependence of a heuristic evaluation function on its other input parameters — the search history.

		History	
		Independent	Dependent
Path	Independent	Classical	h_{LA} (Chapter 4), Selective Max (Chapter 5)
	Dependent	landmark count (Richter et al., 2008)	Future work (Chapter 4)

Table 3.1: Information dependence of some heuristics

Definition 4 (History Independent Heuristic)
Let $\mathbf{P} = \langle S, I, G, A, f, \mathcal{C} \rangle$ *be a search problem, and let* $h : \mathcal{H}_{\mathbf{P}} \times \Gamma \to \mathbb{R}^{0+}$ *be a heuristic evaluation function.* h *is called* history independent *iff for any two histories* $\omega_1, \omega_2 \in \mathcal{H}_{\mathbf{P}}$*, and for any* I*-path* $\pi \in \Gamma$*,* $h(\omega_1, \pi) = h(\omega_2, \pi)$*.*

History independent heuristics do not depend at all on the search history, only on the path they evaluate. With such heuristics we will sometimes abuse notation, and write $h(\pi)$ instead of $h(\omega, \pi)$. If a heuristic is both history independent and path independent, and thus depends only on the state being evaluated, we call it a *classical* heuristic. With classical heuristics we will sometimes abuse notation and write $h(s)$ instead of $h(\omega, \pi)$ for some path π reaching state s. If a heuristic is not history independent, we call it *history dependent*.

Table 3.1 gives some examples of heuristics, and which types of dependence they exhibit. One special class of path independent, history dependent heuristics are what we call multi-path dependent heuristics. To define multi-path dependent heuristics, we must first define the restriction of the structure $\mathcal{G}_\omega$ explored by history ω to state s, denoted by $\mathcal{G}_\omega|s$, which is the subgraph of $\mathcal{G}_\omega$ containing only edges which lie upon some path from I to s, and their adjacent nodes. Using this concept, we can now define multi-path dependent heuristics:

Definition 5 (Multi-Path Dependent Heuristic)
Let $\mathbf{P} = \langle S, I, G, A, f, \mathcal{C} \rangle$ *be a search problem, and let* $h : \mathcal{H}_{\mathbf{P}} \times \Gamma \to \mathbb{R}^{0+}$ *be a path independent heuristic evaluation function.* h *is called* multi-path dependent *iff for any two histories* $\omega_1, \omega_2 \in \mathcal{H}_{\mathbf{P}}$*, and for any* I*-path* $\pi \in \Gamma$ *such that* $I[\![\pi]\!] = s$*, if* $\mathcal{G}_{\omega_1}|s = \mathcal{G}_{\omega_2}|s$ *then* $h(\omega_1, \pi) = h(\omega_2, \pi)$*.*

In other words, the estimate of a multi-path dependent heuristic for state s only depends on the explored part of the state-space that can reach s — that is, on the discovered paths to s.

3.2.2 Admissibility

The second dimension we consider in our taxonomy is admissibility. A heuristic evaluation function is said to be admissible if it never overestimates the distance from any state to the closest goal to it. Admissible heuristics are important, as there are several heuristic search algorithms which can guarantee that an optimal solution will be found, when using an admissible heuristic.

When the heuristic in question is a classical heuristic, the above verbal definition is the same as the well known inequality: $h(s) \leq h^*(s)$. However, when heuristics can depend on something other than the state they evaluate, things become slightly more complicated. The following definition describes a heuristic which never overestimates the goal distance:

Definition 6 (Admissible Heuristic)
Let $\mathbf{P} = \langle S, I, G, A, f, \mathcal{C} \rangle$ *be a search problem, and let* $h : \mathcal{H}_{\mathbf{P}} \times \Gamma \rightarrow \mathbb{R}^{0+}$ *be a heuristic evaluation function.* h *is called* admissible *iff for any* I*-path* $\pi \in \Gamma$*, and for any search history* $\omega \in \mathcal{H}_{\mathbf{P}}$*,* $h(\omega, \pi) \leq h^*(I[\![\pi]\!])$*.*

Another well known notion is *consistency*. Consistency is typically stated by the following inequality: $h(s) \leq k(s, s') + h(s')$. Adapting this to our formal mathematical framework, we get the following definition:

Definition 7 (Consistent Heuristic)
Let $\mathbf{P} = \langle S, I, G, A, f, \mathcal{C} \rangle$ *be a search problem, and let* $h : \mathcal{H}_{\mathbf{P}} \times \Gamma \rightarrow \mathbb{R}^{0+}$ *be a heuristic evaluation function.* h *is called* consistent *iff for any two* I*-paths* $\pi, \pi' \in \Gamma$*, and for any two search histories* $\omega, \omega' \in \mathcal{H}_{\mathbf{P}}$*,* $h(\omega, \pi) \leq k(I[\![\pi]\!], I[\![\pi']\!]) + h(\omega', \pi')$*.*

While consistency is a stronger property than admissibility (Pearl, 1984), even admissibility is a very strong requirement in itself — it requires that the heuristic does not overestimate the goal distance for *all* states. Recall that there are several heuristic search algorithms which guarantee that an optimal solution will be found when using an admissible heuristic. This does not mean that using an admissible heuristic is a necessary condition – in fact, it has already been noted that heuristics which sometime overestimate the goal distance can still guarantee an optimal solution (Dechter and Pearl, 1985). However, the following definitions, which capture weaker notions than admissibility, have not appeared as properties of heuristic evaluation functions in the literature:

Definition 8 ($\hat{S}$-Admissible Heuristic)
Let $\mathbf{P} = \langle S, I, G, A, f, \mathcal{C} \rangle$ be a search problem, let $\hat{S} \subseteq S$ be a set of states of $\mathbf{P}$, and let $h : \mathcal{H}_{\mathbf{P}} \times \Gamma \to \mathbb{R}^{0+}$ be a heuristic evaluation function. h is called $\hat{S}$-admissible *iff for any I-path π such that $I[\![\pi]\!] \in \hat{S}$, and for any search history $\omega \in \mathcal{H}_{\mathbf{P}}$, $h(\omega, \pi) \leq h^*(I[\![\pi]\!])$.*

Definition 8 requires that h never overestimate the goal distance *only of states in $\hat{S}$*. Thus, S-admissibility is equivalent to admissibility, and when $\hat{S} \subset S$, $\hat{S}$-admissibility is a weaker requirement than admissibility. The following definition captures a special case of $\hat{S}$-admissible heuristics, whose importance we will discuss later:

Definition 9 (Globally Admissible Heuristic)
Let $\mathbf{P} = \langle S, I, G, A, f, \mathcal{C} \rangle$ be a search problem, let $h : \mathcal{H}_{\mathbf{P}} \times \Gamma \to \mathbb{R}^{0+}$ be a $\hat{S}$-admissible heuristic evaluation function. h is called globally admissible *iff there exists some optimal solution $\rho = \langle a_0, \ldots, a_n \rangle$ of $\mathbf{P}$, such that $I \in \hat{S}$ and $\{I[\![\langle a_0, \ldots, a_i \rangle]\!] | 0 \leq i \leq n\} \subseteq \hat{S}$.*

In other words, a heuristic is globally admissible if there exists some optimal path ρ, such that h never overestimates the goal distance from any state along ρ. In fact, some optimality-preserving techniques for search space pruning, such as symmetry breaking and state-space reductions (Fox and Long, 2002; Rintanen, 2003; Coles and Smith, 2008; Chen and Yao, 2009; Pochter et al., 2011), can be seen as using a globally admissible heuristic, assigning a heuristic value of ∞ to some states, despite the fact that the goal is achievable from these states.

In the next section, we discuss how one can use globally admissible heuristics to guarantee that an optimal solution will be found. However, we first present an even weaker notion, which can still be enough to guarantee an optimal solution will be found.

Definition 10 (χ-Admissible Heuristic)
Let $\mathbf{P} = \langle S, I, G, A, f, \mathcal{C} \rangle$ be search problem, let $\chi \subseteq \Gamma$ be a set of I-paths, and let $h : \mathcal{H}_{\mathbf{P}} \times \Gamma \to \mathbb{R}^{0+}$ be a heuristic evaluation function. h is called χ-admissible *if for any prefix π of any path $\rho \in \chi$, and for any search history $\omega \in \mathcal{H}_{\mathbf{P}}$, $h(\omega, \pi) \leq h^*(I[\![\pi]\!])$.*

In other words, a χ-admissible heuristic assigns admissible estimates to prefixes of any path in χ. We now present two special cases of χ-admissible heuristics:

Definition 11 (Path and Globally Path Admissible Heuristic) *Let* $\mathbf{P} = \langle S, I, G, A, f, \mathcal{C} \rangle$ *be a solvable search problem, and let* $h : \mathcal{H}_{\mathbf{P}} \times \Gamma \rightarrow \mathbb{R}^{0+}$ *be a* χ*-admissible heuristic evaluation function.*

- h *is called* path admissible *iff* χ *contains* all *optimal solutions of* $\mathbf{P}$*.*
- h *is called* globally path admissible *iff* χ *contains* at least one *optimal solution of* $\mathbf{P}$*.*

While path admissibility is a requirement over all prefixes of *all* optimal solutions, global path admissibility is limited to prefixes of *some* optimal solution. The requirement that $\mathbf{P}$ is solvable is necessary here, as otherwise it is not possible to discuss optimal solutions of $\mathbf{P}$. Note that overestimation in unsolvable problems is meaningless, as the goal distance from any state is ∞. As we will see in Section 3.3 it is possible to guarantee that an optimal solution will be found using a globally path admissible heuristic.

3.3 Search Algorithms

While heuristics are interesting in and of themselves, their purpose is to guide a *search algorithm*. In this work we are concerned with forward search for optimal solutions, which immediately brings to mind the classic A^* search algorithm (Hart et al., 1968). A^* is known to find an optimal solution, when using an admissible heuristic. However, admissibility is a strong requirement, making demands of the heuristic estimates of *all* states. In fact, as we will prove later, the weaker notion of a globally admissible heuristic is enough to guarantee that A^* will find an optimal solution.

However, A^* is adapted to classical heuristics, and does not exploit non-classical heuristics to their full extent. In this section, we introduce some search algorithms which are better suited to non-classical heuristics than A^*. For completeness of presentation, we first present the A^* search algorithm, in terms of our mathematical framework. Algorithm 1 presents the pseudo-code of A^*, with the search history ω and the paths to generated states made explicit. In order to avoid confusion with the heuristic evaluation function h, we use the notation $\overline{h}(s)$ to denote the heuristic value currently associated with state s, as well as $\overline{g}(s)$ to denote the cost of the currently best known path to s, and $\overline{f}(s) := \overline{g}(s) + \overline{h}(s)$. $Pa(s)$ stores the current parent of state s. We also use the function $trace(s)$, which returns the I-path to s along the current parent pointers. It is easy to see that $trace(s)$ is well-defined for any generated state.

Algorithm 1 A^*

```
1  Closed ← ∅, Open ← ∅
2  ω ← ⟨⟩
3  ḡ(I) ← 0, h̄(I) ← h(ω, trace(I)), f̄(I) ← ḡ(I) + h̄(I)
4  Open.insert(I)
5  while Open ≠ ∅ do
6      remove s with minimum f̄(s) from Open, break ties in favor of low h̄(s)
7      if IS_GOAL(s) then
8        return trace(s)
9      end if
10     Closed.insert(s)
11     ω.append(s)
12     for ⟨a, s'⟩ ∈ succ^a(s) do
13       if s' ∉ Closed ∪ Open then
14         ḡ(s') ← ḡ(s) + C(a), Pa(s') ← ⟨s, a⟩
15         h̄(s') ← h(ω, trace(s'))
16         f̄(s') ← ḡ(s') + h̄(s')
17         Open.insert(s')
18       else if ḡ(s) + C(a) < ḡ(s') then
19         ḡ(s') ← ḡ(s) + C(a), Pa(s') ← ⟨s, a⟩
20         f̄(s') ← ḡ(s') + h̄(s')
21         Open.insert(s')
22       end if
23     end for
24 end while
25 return NO SOLUTION
```

We will not go over the entire pseudo-code of A^*, but simply point out where the search history and the paths are made explicit here. Note that it is not necessary to explicitly keep track of the search history, nor to use the *trace* function whenever a state is evaluated, when implementing A^*. Rather, we use the variable ω to illustrate where the search history is modified and used in A^*, and call *trace* in heuristic computation to illustrate that the estimate depends on the current path. Line 2 initializes the search history ω to an empty list. In line 11 the state that is currently being expanded is added to the history. The history and the *trace* function are both used in the heuristic computation in lines 3 and 15.

To show that A^* is guaranteed to find an optimal solution given a globally admissible heuristic, we follow the same proof by Pearl (1984). While this proof is not new, it was originally used to show that A^* with an *admissible* heuristic is guaranteed to find an optimal solution. Furthermore, we will use almost the exact same proof to show that the new search algorithms we propose are also guaranteed to find optimal solutions. We begin by proving the "A^* Lemma":

Lemma 1
At any time before A^* terminates, there exists on the open list a state s which lies on some optimal solution ρ, with $\overline{f}(s) \leq c^*$

Proof: Assume the states which lie along ρ are $\langle I, s_1, \ldots, s, \ldots, s_n \rangle$, and let s be the shallowest state on ρ which is open (there is at least one, because the goal state s_n is only closed when A^* terminates). Since all ancestors of s along ρ are closed, and the path $\langle I, s_1, \ldots, s \rangle$ is optimal, it must be that $\overline{g}(s) = g^*(s)$. h is globally admissible, and so it returns admissible estimates for any state along an optimal path. Because s lies on an optimal solution:

$$\overline{f}(s) = \overline{g}(s) + \overline{h}(s) = g^*(s) + \overline{h}(s) \leq g^*(s) + h^*(s) = c^*$$

■

The following Theorem shows why Lemma 1 implies that A^* is guaranteed to find an optimal solution:

Theorem 1 *A^* with a globally admissible heuristic returns an optimal solution*

Proof: Suppose to the contrary that A^* terminates with a goal state t for which $\overline{f}(t) = \overline{g}(t) > c^*$. A^* inspects states for compliance with the

termination condition only after it selects them for expansion. Hence, when t was selected for expansion, it satisfied:

$$\forall s \in Open : \overline{f}(t) \leq \overline{f}(s)$$

This means that, immediately prior to termination, every state s' in the open list satisfied $\overline{f}(s') > c^*$. This, however, contradicts Lemma 1 which guarantees the existence of at least one state in the open list with $\overline{f}(s) \leq c^*$. Therefore, the terminating t must have $\overline{g}(t) = c^*$, which means that A^* found an optimal solution. ■

While A^* has been shown to be efficient when using admissible classical heuristics (Dechter and Pearl, 1985), it does not do so well with non-classical heuristics. The main reason that A^* is not suited to non-classical heuristics, is that it computes the heuristic estimate for state s only the first time s is generated (line 15). Whenever a state is generated again, A^* simply uses $\overline{h}(s)$ (line 20), which contains the heuristic value which was computed for s previously. When the heuristic h is non-classical, its heuristic estimate can change. If h is history dependent, the heuristic estimate for state s can change any time. However, if h is path dependent or multi-path dependent, we know when these changes can occur — when a new path to state s is discovered. We therefore propose a modified version of A^* which better exploits path dependent and multi-path dependent heuristics.

3.3.1 Exploiting Path and Multi-Path Dependence

The first observation we make here is that any admissible path dependent heuristic h can induce a better informed admissible multi-path dependent heuristic h', by taking h' as the maximum over all known paths. Abusing notation as before, we can write $h'(\omega, s) := \max_{\pi \in \mathcal{G}_\omega|s} h(\omega, \pi)$, where $\pi \in \mathcal{G}_\omega|s$ means that π is an I-path reaching state s in $\mathcal{G}_\omega|s$. Therefore, we focus on the case of multi-path dependent heuristics, and show how to better exploit them.

Recall that A^* evaluates each state only once, the first time it is generated. Thus, if we want our new search algorithm to behave differently from, and hopefully outperform, A^*, it is clear we must recompute heuristic estimates for the same state. However, the question of when to perform this reevaluation requires some discussion. There's no point in reevaluating some state s unless its heuristic estimate *might* change. For multi-path dependent heuristics, this can happen only if $\mathcal{G}_\omega|s$ has changed, which can only happen when some state s' is expanded, and s' has a successor in $\mathcal{G}_\omega|s$.

Assume we have expanded some state s', and generated state s, which has already been generated. This means we have discovered at least one new path to s, as well as to all descendants of s. The question now is, which states do we reevaluate. One option, which we will call the eager option, is to immediately reevaluate the heuristic value of s, and also all of the descendants of s. A slightly less eager option is to update only those descendants that are within a limited distance from s. A similar question has been explored in the context of bidirectional pathmax (BPMX) updates (Zhang et al., 2009), where distance-limited updates were proposed and empirically examined. These experiments showed there is no clear-cut best choice about how to propagate this information.

However, typically in domain-independent planning, heuristic computation is much more expensive than state expansion. We therefore try to reduce the number of heuristic reevaluations, and only perform them when this is deemed necessary. Thus, the approach we take in the MPD-A* search algorithm (Karpas and Domshlak, 2009), originally called LM-A^*, is to be as lazy as possible. MPD-A* exploits the fact that it is possible that many different paths to s will be discovered while s is in the open list. Thus, reevaluating the heuristic for s *every time* a new path is discovered is rather wasteful. Instead, it is enough to check whether the heuristic estimate for s increased only when it is really needed — when s is removed from the open list. Only then is s reevaluated, and if its heuristic estimate increased, another state is selected for expansion instead.

The pseudo-code for MPD-A* is given in Algorithm 2. To illustrate that MPD-A* is meant for multi-path dependent heuristics, we write $h(\mathcal{G}_\omega|s)$ instead of $h(\omega, \pi)$, as the heuristic estimate for state s with history ω only depends on $\mathcal{G}_\omega|s$. MPD-A* is based on the A^* search algorithm, with a few modifications: MPD-A* associates a *dirty* bit with each known state. A state s is dirty when a new path to it has been discovered, but this new path is not yet reflected in the stored heuristic estimate $\overline{h}(s)$ (line 21). Whenever $\overline{h}(s)$ is updated, the dirty bit associated with s is cleared (lines 3, 11 and 24). Before a state s is removed from the open list, its dirty bit is first checked (line 10). If s is dirty, then it is reevaluated using h (line 11), and only if $h(s)$ did not increase, is s really removed from the open list. If the heuristic estimate of s did increase, then $\overline{h}(s)$ and $\overline{f}(s)$ are updated, s is put back into the open list with these new values, and another state is chosen from the open list (lines 13 – 15). To see that MPD-A* is guaranteed to find an optimal solution, the same proof that A^* finds an optimal solution, given in Lemma 1 and Theorem 1, works.

Finally, we remark that although MPD-A* is stated in terms of storing

Algorithm 2 MPD-A*

```
1  Closed ← ∅, Open ← ∅
2  ω ← ⟨⟩
3  ḡ(I) ← 0, h̄(I) ← h(G_ω|I), f̄(I) ← ḡ(I) + h̄(I), dirty(I) ← FALSE
4  Open.insert(I)
5  while Open ≠ ∅ do
6      remove s with minimum f̄(s) from Open, break ties in favor of low h̄(s)
7      if IS_GOAL(s) then
8        return trace(s)
9      end if
10     if dirty(s) then
11       new_h ← h(G_ω|s), dirty(s) ← FALSE
12       if new_h > h̄(s) then
13         h̄(s) ← new_h, f̄(s) ← ḡ(s) + h̄(s)
14         Open.insert(s)
15         continue
16       end if
17     end if
18     Closed.insert(s)
19     ω.append(s)
20     for ⟨a, s'⟩ ∈ succ^a(s) do
21       dirty(s') ← TRUE
22       if s' ∉ Closed ∪ Open then
23         ḡ(s') ← ḡ(s) + C(a), Pa(s') ← ⟨s, a⟩
24         h̄(s') ← h(G_ω|s'), dirty(s') ← FALSE
25         f̄(s') ← ḡ(s') + h̄(s')
26         Open.insert(s')
27       else if ḡ(s) + C(a) < ḡ(s') then
28         ḡ(s') ← ḡ(s) + C(a), Pa(s') ← ⟨s, a⟩
29         f̄(s') ← ḡ(s') + h̄(s')
30         Open.insert(s')
31       end if
32     end for
33 end while
34 return NO SOLUTION
```

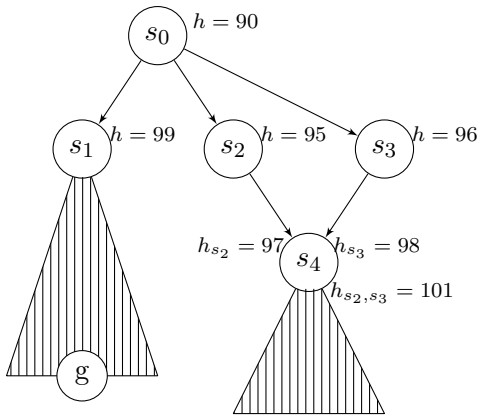


Figure 3.1: Illustration of state space for MPD-A* example

the search history ω, there is no need to keep track of it explicitly. Rather, it is enough to store only the data which the heuristic h actually needs for the heuristic computation, which could be much more compact in some cases.

The following example, depicted in Figure 3.1, illustrates the difference between A^* and MPD-A*. The initial state s_0 has three successors: s_1, s_2, and s_3, and a solution is only present under s_1 (denoted by g). s_2 and s_3 share a successor, s_4. The heuristic in this example, h, is multi-path dependent. Since there is only a single path to s_0, s_1, s_2, and s_3, we simply list their h values: $h(s_0) = 90$, $h(s_1) = 99$, $h(s_2) = 95$, and $h(s_3) = 96$. There are two possible paths to state s_4, and so we must list three different heuristic values for s_4 — one for each possible path, and one for their combination. The heuristic value assigned to s_4 when using the path through s_2 is $h_{s_2}(s_4) = 97$, the heuristic value using the path through s_3 is $h_{s_3}(s_4) = 98$, and the heuristic value using both paths is $h_{s_2,s_3}(s_4) = 101$. All actions in this example are unit cost.

The A^* algorithm will first expand s_0, generating s_1, s_2, and s_3. The next state to be expanded is the state with the lowest $\overline{f}$ value, which is s_2, with $\overline{f}(s_2) = 96$. After expanding s_2, s_4 is generated, with $\overline{h}(s_4) = 97$. Because A^* evaluates each state only the first time it is reached, this estimate will never change. The next state to be expanded will be s_3, followed by s_4, and then the entire subtree under s_4 could be exhausted before s_1 is tried.

On the other hand, MPD-A* will start like A^* by expanding s_0 and then s_2, generating s_4 with $\overline{h}(s_4) = 97$. s_3 will be expanded next, and generate s_4 again, marking s_4 as dirty. The next state to be removed from the open list is

s_4, as it was initially put into the open list with $\overline{h}(s_4) = 97$. However, when it is removed from the open list, its dirty bit is checked, and because a new path to s_4 was found since the last heuristic evaluation, the new heuristic value is computed, updating $\overline{h}(s_4)$ to 101. Now s_1 looks more promising, so it will be expanded next, finding a solution without expanding the subtree under s_4.

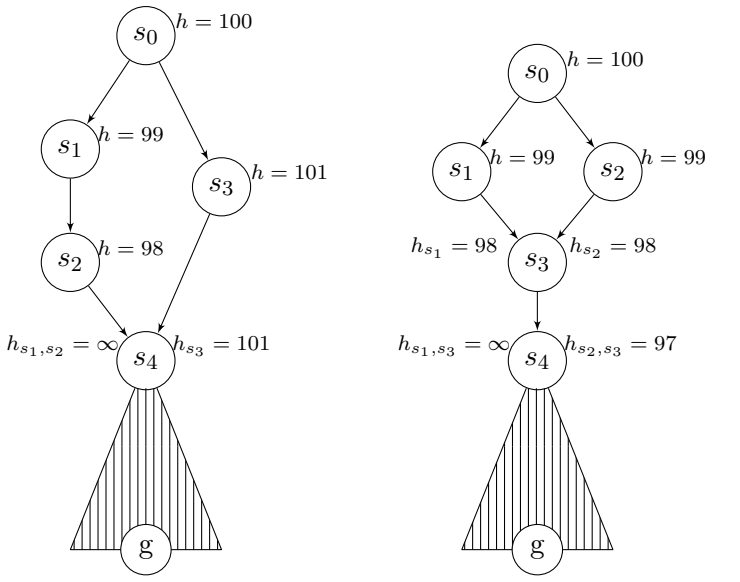
The advantage of MPD-A* over A^* in this example depends on the fact that a new path to s_4 was discovered, while s_4 was in the open list. As A^* evaluates each state only *before* it is inserted into the open list, it can not use the extra information that the new path to s_4 provides. In the empirical evaluation in Chapter 6, we will show that the above example is not just some artificial example, and MPD-A* does perform better than A^* on actual benchmarks from the International Planning Competition, using a real multi-path dependent heuristic which we describe in Chapter 4.

While MPD-A* is suited for multi-path dependent admissible heuristics, we have also introduced more relaxed versions of admissibility. As previously stated A^* search using a globally admissible heuristic will find an optimal solution. The same is true for MPD-A*— given a multi-path dependent globally admissible heuristic, it will find an optimal solution. However, for χ-admissible heuristics, we must use other search algorithms, which will be discussed next.

3.3.2 Exploiting χ-Admissible Heuristics

While MPD-A* does a good job of exploiting admissible path and multi-path dependent heuristics, we would also like to be able to exploit χ-admissible heuristics. Unfortunately, even using a path admissible heuristic with A^* (as well as with MPD-A*) search does not guarantee that an optimal solution will be found. The reason for this is that both A^* and MPD-A* assume that any heuristic estimate which is generated by the heuristic h is admissible, which is not the case with path admissible heuristics, when using a suboptimal path to s to evaluate the distance from s to the goal.

Figure 3.2(a) illustrates the state-space for such an example. The initial state s_0 has two successors: s_1 and s_3. s_1 has a single successor s_2, and s_2 and s_3 both share a common successor — s_4. The heuristic h in this example is history independent and path dependent. Since there is a single path to s_0, s_1, s_2 and s_3, we simply list their heuristic values: $h(s_0) = 100$, $h(s_1) = 99$, $h(s_2) = 98$, and $h(s_3) = 101$. There are two paths to s_4: through s_1 and s_2, or through s_3. The heuristic estimate of the first path, denoted $h_{s_1,s_2}(s_4)$ is ∞ — a dead end. The heuristic estimate of the second path,



(a) Path admissible (b) Globally path admissible

Figure 3.2: Illustration of state spaces

denoted $h_{s_3}(s_4)$ is 101. Note that as the only optimal solution goes through s_3, h is path admissible. However, A^* would first expand s_1 and s_2, as they have lower f-values than s_3. Thus s_4 would first be generated with the path going through s_1 and s_2, and s_4 would be declared a dead-end, rendering the search incomplete.

The reason for this "problem" is that, with a χ-admissible heuristic, when given two paths to the same state, it's no longer possible to irrevocably discard one of them, and still guarantee optimality. Although in the previous example it was possible to distinguish between the two possible paths by their cost, this is not always true, as illustrated by the example depicted in Figure 3.2(b). There are two paths to state s_4, the first going through s_1, s_3, and the second going through s_2, s_3. The heuristic h in this example is globally path admissible — it yields heuristic estimates only for the latter path, and not for the former. Upon generating s_3, both paths look the same, both in terms of g and h values. Hence, there is no reason to prefer to choose s_2 over s_1 as the parent of s_3. However, if we choose s_1 as the parent of s_3, then when generating s_4, we would get a heuristic estimate of ∞ — a dead end. Thus, if we want to guarantee that an optimal solution will

be found using a globally path admissible heuristic, we can not make what Dechter and Pearl (1985) call *irrevocable parent selection* decisions. This implies that we must be able to maintain several different paths to the same state as candidates, which would suggest we must use a tree-based search algorithm.

However, if the heuristic h is path admissible, providing us with admissible estimates for all prefixes of all optimal solutions, we do not have to resort to tree-based search. In this case, we can make an irrevocable parent selection decision based on g alone — a cheaper path to state s will always be better than a more expensive path. In case we have two paths reaching the same state with the same cost, we know that either both paths will yield admissible estimates (if they are both optimal), or that both paths are not guaranteed to yield admissible estimates. In either case, we can choose any of these paths without compromising the optimality of the solution that will be returned. Thus, a simple modification of A^*, which recomputes the heuristic estimate *every* time a state is reached via a cheaper path, will guarantee that an optimal solution is found. The pseudo code for this is found in Algorithm 3.

The only difference between *path-*A^* and A^* is in line 20, which is executed after a cheaper path to a known state s' is found. A^* simply uses the previous heuristic estimate for s'. On the other hand, *path-*A^* computes a new heuristic estimate (based on the new path), and then, like A^*, computes a new f-value according to the new path and the new heuristic estimate.

To see that *path-*A^* using a path admissible heuristic is guaranteed to find an optimal solution, we prove a variant of Lemma 1 for *path-*A^*:

Lemma 2
At any time before path-A^* *terminates, there exists on the open list a state* s *which lies on some optimal plan* ρ*, with* $\overline{f}(s) \leq c^*$

Proof: Assume the states which lie along ρ are $\langle I, s_1, \ldots, s, \ldots, s_n \rangle$, and let s be the shallowest state on ρ which is open (there is at least one, because the goal state s_g is only closed when *path-*A^* terminates). Since all ancestors of s along ρ are closed, and the path $\langle I, s_1, \ldots, s \rangle$ is optimal, it must be that $\overline{g}(s) = g^*(s)$. Using the admissibility of h along prefixes of any optimal solution, including ρ, and the fact that s lies on an optimal solution, we obtain:

$$\overline{f}(s) = \overline{g}(s) + \overline{h}(s) = g^*(s) + \overline{h}(s) \leq g^*(s) + h^*(s) = c^*$$

■

Algorithm 3 *path-A**

```
1  Closed ← ∅, Open ← ∅
2  ω ← ⟨⟩
3  ḡ(I) ← 0, h̄(I) ← h(ω, trace(I)), f̄(I) ← ḡ(I) + h̄(I)
4  Open.insert(I)
5  while Open ≠ ∅ do
6     remove s with minimum f̄(s) from Open, break ties in favor of low h̄(s)
7     if IS_GOAL(s) then
8       return trace(s)
9     end if
10    Closed.insert(s)
11    ω.append(s)
12    for ⟨a, s'⟩ ∈ succ^a(s) do
13       if s' ∉ Closed ∪ Open then
14          ḡ(s') ← ḡ(s) + C(a), Pa(s') ← ⟨s, a⟩
15          h̄(s') ← h(ω, trace(s'))
16          f̄(s') ← ḡ(s') + h̄(s')
17          Open.insert(s')
18       else if ḡ(s) + C(a) < ḡ(s') then
19          ḡ(s') ← ḡ(s) + C(a), Pa(s') ← ⟨s, a⟩
20          h̄(s') ← h(ω, trace(s'))
21          f̄(s') ← ḡ(s') + h̄(s')
22          Open.insert(s')
23       end if
24    end for
25 end while
26 return NO SOLUTION
```

Using this lemma, the proof of Theorem 1 also shows that *path-*A^* with a path admissible heuristic is guaranteed to find an optimal solution.

When using a globally path admissible heuristic, the above algorithm is not enough to guarantee optimality. The example state space in Figure 3.2(b) already illustrates this — if we reach s_3 via s_1 first, we would not recompute a new heuristic estimate for s_3 after expanding s_2. However, given some more knowledge about a globally path admissible heuristic, it is sometimes possible to use a similar algorithm to find an optimal solution. We present an example of this in Section 4.4.

This chapter has been devoted to laying the mathematical framework for non-classical heuristics. In the rest of this work, we will describe some non-classical heuristics for domain-independent planning.

Chapter 4

Landmark Based Heuristics

We now turn our attention back to domain independent planning, and present the concept of *landmarks*, and landmark based heuristics. We begin by reviewing the definitions of landmarks and orderings. Then we briefly discuss some landmark detection techniques. Finally we discuss how to construct landmark based heuristics, and the search techniques which are suitable for using them.

4.1 Landmarks

The definitions we use here follow Domshlak et al. (2012). The first definition we need is that of a plan trace:

Definition 12 (Trace)
Let $\Pi = \langle P, I, G, A, \mathcal{C} \rangle$ *be a planning task, and let* π *be a plan for* Π*. The* trace $tr(\pi)$ *of* $\pi = \langle a_1, \ldots, a_n \rangle$ *is given by the sequence of propositional formulas* $\langle tr_1, \ldots, tr_n, tr_{n+1} \rangle$ *over the set of boolean variables* $P \cup A$*, constructed as follows, where states are interpreted as conjunctions of the facts they contain:*

- $tr_1 = I \wedge a_1$
- $tr_i = I[\![\langle a_1, \ldots, a_{i-1} \rangle]\!] \wedge a_i$ *for* $i = 2, \ldots, n$
- $tr_{n+1} = I[\![\langle a_1, \ldots, a_n \rangle]\!]$

For simplicity, we write tr rather than $tr(\pi)$ whenever π is clear from the context. We further introduce the notion of *occurrences* of a formula in a trace step: we denote by $occ(tr(\pi), \phi)$ (or $occ(\phi)$ when π is clear from

the context) the set of step indices in tr that entail the formula ϕ, i. e. $occ(\phi) := \{i \mid tr_i \models \phi\}$. We use a special notation for the first time ϕ occurs: $first(\phi) := \min(occ(\phi))$. We also use shorthand notation for the indices of the states where ϕ *becomes* true, i. e. $bt(\phi) := \{i | i \in occ(\phi), i-1 \notin occ(\phi)\}$, with special notation for the last time ϕ becomes true: $last(\phi) := \max(bt(\phi))$. We now define landmarks and orderings based on the occurrences of formulas in plan traces.

Definition 13 (Landmark)
A landmark *for a planning task* Π *is a propositional formula* ϕ *over the facts and actions of* Π *such that for every plan* π *for* Π, $occ(\phi) \neq \emptyset$.

In other words, a landmark for Π is a logical formula that is entailed by at least one step during the execution trace of every plan for Π. Restrictions on the form of ϕ result in various types of landmarks, such as *fact* landmarks, consisting of a single fact, *action* landmarks consisting of an action, and *disjunctive* and *conjunctive* (action) landmarks, consisting of a disjunction or conjunction of facts or actions, respectively.

An *ordering* between two landmarks is a statement about the indices of the steps that satisfy these landmarks in the trace of all plans. Various types of orderings can be defined (Hoffmann et al., 2004):

Definition 14 (Landmark Orderings) *Let* ϕ_1 *and* ϕ_2 *be two landmarks for a planning task* Π.

Natural *There is a* natural ordering $\phi_1 \ll_{\mathrm{n}} \phi_2$ *iff for all plans* π *of* Π, $first(\phi_1) < first(\phi_2)$.

Necessary *There is a* necessary ordering $\phi_1 \ll_{\mathrm{nec}} \phi_2$ *iff for all plans* π *of* Π, $i \in bt(\phi_2) \Rightarrow i-1 \in occ(\phi_1)$.

Greedy Necessary *There is a* greedy-necessary ordering $\phi_1 \ll_{\mathrm{gn}} \phi_2$ *iff for all plans* π *of* Π, $(first(\phi_2)-1) \in occ(\phi_1)$.

A *natural* ordering $\phi_1 \ll_{\mathrm{n}} \phi_2$ exists if every plan makes ϕ_1 true before it makes ϕ_2 true for the first time. A *necessary* ordering $\phi_1 \ll_{\mathrm{nec}} \phi_2$ exists if, for every plan π and every time π makes ϕ_2 true, ϕ_1 is always true in the trace step *immediately before* ϕ_2 becomes true. A *greedy-necessary* ordering $\phi_1 \ll_{\mathrm{gn}} \phi_2$ exists if, for every plan, ϕ_1 is true in the trace step *immediately before* the plan makes ϕ_2 true for the *first* time.

It is easy to see from the definitions that for any two landmarks ϕ_1 and ϕ_2:

$$\phi_1 \ll_{\mathrm{nec}} \phi_2 \Longrightarrow \phi_1 \ll_{\mathrm{gn}} \phi_2 \Longrightarrow \phi_1 \ll_{\mathrm{n}} \phi_2$$

We conclude the definitions of landmarks and orderings by remarking that other types of orderings — namely, reasonable ordering — have been proposed (Hoffmann et al., 2004). However, their definition is somewhat involved, and we do not exploit them in this work, so we omit their exact definition.

The definitions of landmarks and orderings refer to all solutions of a planning task. Thus, as can be expected, even determining whether a single fact is a landmark or not is PSPACE-hard (Hoffmann et al., 2004). Nonetheless, some sound polynomial-time methods for detecting landmarks and orderings have been proposed (Zhu and Givan, 2003; Richter et al., 2008; Keyder et al., 2010).

While the details of these detection techniques are not relevant to the rest of this work, it is important to note that landmark detection is more computationally expensive that most heuristics for domain-independent planning. Therefore, landmark based heuristics typically discover landmarks and orderings in a pre-search phase, building a *landmark graph*. The landmark graph is an edge-labeled graph, whose nodes are the discovered landmarks, with edges corresponding to the discovered orderings, labeled according to the ordering type. For the remainder of this chapter, we assume that a set of landmarks LM and orderings $\ll$ have been discovered during a pre-search phase. In the next section, we will explain how to derive heuristics based on this landmark graph.

4.2 Heuristics based on Landmark Graphs

Landmarks were first used as a source of heuristic information by Richter et al. (2008), and later in the LAMA planner (Richter and Westphal, 2010). The heuristic estimate of state s is "the number of landmarks that remain to be achieved from state s onwards". As all the landmarks of a task must necessarily be achieved along any plan, we can judge the proximity of a state to the goal by counting how many landmarks still need to be achieved. Note that this heuristic is not admissible, as a single action may achieve more than one landmark.

One option for calculating the landmarks that need to be achieved from state s onwards is to use a landmark discovery procedure for s (treating s as the initial state for the computation). However, as previously mentioned, this would result in a computationally expensive heuristic. Another, less computationally intensive option, is to compute the landmarks for the initial state of the task once, and account for which landmarks from this set have

already been achieved on the way to s. Using this option means that the landmark heuristic is no longer a classical, state dependent heuristic, but rather a path dependent heuristic, as it depends on the path by which s was reached. This is the method used in LAMA, where the set of landmarks that need to be achieved after reaching state s via path π is defined as

$$LM(s, \pi) := \big(LM \setminus Accepted(s, \pi)\big) \cup ReqAgain(s, \pi),$$

where LM is the set of prediscovered landmarks, $Accepted(s, \pi) \subseteq LM$ is the set of landmarks that have been achieved along path π (note that once a landmark has been achieved, it remains accepted, even if it is made false afterwards), and $ReqAgain(s, \pi) \subseteq Accepted(s, \pi)$ is the set of landmarks that are *required again*. A landmark ϕ is required again if it has previously been achieved, but is not true in the current state and must be achieved again.

Since it is computationally hard to determine if a landmark ϕ must be achieved again, LAMA uses a sound approximation to determine whether $\phi \in ReqAgain(s, \pi)$, captured by two rules: (i) ϕ does not hold in s, and (ii) ϕ is a top-level goal, or there exists some other landmark $\phi' \notin Accepted(s, \pi)$, and $\phi \ll_{\text{gn}} \phi'$. These rules capture the case where ϕ has been achieved along path π, but has been made false, and we can prove that ϕ must be true again at some point, either because it is a top-level goal (and thus must hold at the end) or it is ordered greedy-necessarily before some other landmark ϕ' which has not been achieved (and thus ϕ must hold in the state before ϕ' is achieved). Other approximations for the required-again landmarks have been proposed (Buffet and Hoffmann, 2010), but the exact approximation of the required-again landmarks is orthogonal to their use. In the next section, we will discuss the non-classical nature of landmark-based heuristics. However, for the remainder of this section, we will simply assume that the set of landmarks that need to be achieved, $LM(s, \pi)$, is given.

4.2.1 Admissible Landmark-Based Heuristics

Recall that the number of landmarks that need to be achieved is not an admissible estimate due to the fact that a single action may achieve more than one landmark. However, it is possible to use the framework of additive composition of abstraction heuristics (Katz and Domshlak, 2010a), to obtain an admissible heuristic estimate. This framework is described by the following theorem from Katz and Domshlak (2010a):

Theorem 2 (Action Cost Partitioning) *Let $\Pi, \Pi_1, \ldots, \Pi_k$ be planning tasks, identical except for the operator costs $\mathcal{C}, \mathcal{C}_1, \ldots, \mathcal{C}_k$. Let $\{h_i\}_{i=1}^k$ be a set of* arbitrary *admissible heuristic functions for $\{\Pi_i\}_{i=1}^k$, respectively. If $\mathcal{C}(a) \geq \sum_{i=1}^k \mathcal{C}_i(a)$ for all operators a, then $\sum_{i=1}^k h_i$ is an admissible heuristic for Π.*

Applying this theorem to landmarks is straightforward — each landmark defines a planning task with a heuristic indicating whether the landmark is achieved or not (Helmert and Domshlak, 2009). For this purpose, it is best to consider only action landmarks, where we represent a propositional landmark ϕ by a disjunctive action landmark consisting of the set of relevant achievers (that is, the actions which might achieve ϕ). If ϕ has not been achieved yet (that is, $\phi \notin \mathit{Accepted}(s, \pi)$), these relevant actions are restricted to the first achievers of ϕ, since they are the only actions that may achieve ϕ for the *first* time. If ϕ has already been achieved but is required again, the relevant achievers are all possible achievers of ϕ (that is, all actions which could make ϕ true). Note that although it is computationally hard to find the first achievers of a landmark, it is possible to find an overapproximation of the first achievers (Porteous and Cresswell, 2002), which would still result in an admissible heuristic.

By $\mathit{ach}(\phi \mid s, \pi)$ we denote the relevant achievers of landmark ϕ, and by $L(a \mid s, \pi) := \{\phi \mid a \in \mathit{ach}(\phi \mid s, \pi), \phi \in \mathit{LM}(s, \pi)\}$ the set of landmarks that need to be achieved and are achieved by action a. It is easy to see that if ϕ is a landmark, then $\mathit{ach}(\phi \mid s, \pi)$ forms a disjunctive action landmark (because any plan which achieves ϕ must do so using one of its achievers — $\mathit{ach}(\phi \mid s, \pi)$). Formulating this explicitly gives us the following constraints:

$$\begin{aligned} \forall a \in A: \ \mathcal{C}(a) &\geq \sum_{\phi \in L(a|s,\pi)} \mathcal{C}_\phi(a) \\ \forall \phi \in \mathit{LM}(s, \pi): \ \mathit{cost}(\phi) &\leq \min_{a \in \mathit{ach}(\phi|s,\pi)} \mathcal{C}_\phi(a) \end{aligned} \tag{4.1}$$

Where $\mathcal{C}_\phi$ denotes the action costs in the abstraction corresponding to landmark ϕ. These constraints capture that (i) each action divides at most its own cost between the landmarks it achieves — the condition from Theorem 2, and (ii) each landmark is assigned a cost which is at most the cheapest cost assigned to it by any action — an admissible heuristic estimate for a single landmark.

Following Katz and Domshlak (2010a), given costs which obey these constraints, and abusing notation since this heuristic is history independent,

$h_{LA}(\pi) := \sum_{\phi \in LM(s,\pi)} cost(\phi)$ is an admissible heuristic estimate. Furthermore, these constraints can be expressed as a Linear Program (LP), which, together with the objective of maximizing the total heuristic estimate $h_{LA}(\pi)$, yields an optimal cost partitioning. Recent work has shown that a higher heuristic estimate can be gotten by solving a hitting set problem (Bonet and Helmert, 2010). However, as solving the hitting set problem is NP-hard, this is not a feasible option in practice.

Although the formulation in Equation 4.1 is easy to explain in terms of additive composition of abstractions, it is rather inefficient in terms of the number of variables it requires. This is important if an LP solver is used to find an optimal cost partitioning. The following formulation, which is due to Helmert (2009a) and was used by Keyder et al. (2010), only uses the $cost(\phi)$ variables:

$$\forall a \in A : \sum_{\phi \in L(a|s,\pi)} cost(\phi) \leq \mathcal{C}(a) \tag{4.2}$$

In words, the sum of costs of the landmarks achieved by action a must not exceed the cost of action a. As the following theorem demonstrates, the formulations in Equations 4.1 and 4.2 are equivalent, in the sense that they both define the same set of feasible solutions over the $cost(\phi)$ variables.

Theorem 3 (Constraint Equivalence) *The following two claims hold:*

1. *Let $cost(\phi)$, $\mathcal{C}_\phi(a)$ be costs that satisfy the constraints in Equation 4.1. Then $cost(\phi)$ satisfy the constraints in Equation 4.2.*
2. *Let $cost(\phi)$ be costs that satisfy the constraints in Equation 4.2. Then there exist costs $\mathcal{C}_\phi(a)$ such that $cost(\phi)$, $\mathcal{C}_\phi(a)$ satisfy the constraints in Equation 4.1.*

Proof:

To prove 1, let $cost(\phi)$, $\mathcal{C}_\phi(a)$ be costs that satisfy the constraints in Equation 4.1, and let a be some action. From Equation 4.1 we have that:

$$\sum_{\phi \in L(a|s,\pi)} cost(\phi) \leq \sum_{\phi \in L(a|s,\pi)} \min_{a' \in ach(\phi|s,\pi)} \mathcal{C}_\phi(a')$$

By definition $\phi \in L(a \mid s, \pi) \Leftrightarrow a \in ach(\phi \mid s, \pi)$, therefore $\min_{a' \in ach(\phi|s,\pi)} \mathcal{C}_\phi(a') \leq \mathcal{C}_\phi(a)$ (since a is part of the minimum), so:

$$\sum_{\phi \in L(a|s,\pi)} \min_{a' \in ach(\phi|s,\pi)} \mathcal{C}_\phi(a') \leq \sum_{\phi \in L(a|s,\pi)} \mathcal{C}_\phi(a)$$

And from Equation 4.1:

$$\sum_{\phi \in L(a|s,\pi)} \mathcal{C}_\phi(a) \leq \mathcal{C}(a)$$

thus proving 1.

To prove 2, let $cost(\phi)$ be costs that satisfy the constraints in Equation 4.2, and set $\mathcal{C}_\phi(a) = cost(\phi)$ for all actions a and landmarks ϕ. We will see that these costs satisfy the constraints in Equation 4.1.

From Equation 4.2,

$$\mathcal{C}(a) \geq \sum_{\phi \in L(a|s,\pi)} cost(\phi) = \sum_{\phi \in L(a|s,\pi)} \mathcal{C}_\phi(a)$$

Because of the way we chose $\mathcal{C}_\phi(a)$,

$$cost(\phi) = \min_{a \in ach(\phi|s,\pi)} \mathcal{C}_\phi(a)$$

So the constraints of Equation 4.1 are satisfied.

■

The equivalence shown in Theorem 3 shows that the two formulations define the same set of feasible solution (when restricted to the $cost(\phi)$ variables). Since we know from Theorem 2 that any solution which obeys the constraints in Equation 4.1 gives an admissible estimate, the same holds for any solution for Equation 4.2. Furthermore, when trying to find an optimal cost partitioning, the target function is the same for both — maximize $\sum_{\phi \in LM(s,\pi)} cost(\phi)$ — so both formulations will give the same value under optimal cost partitioning.

4.2.2 Cost-Partitioning Schemes

As already stated, an optimal (as in, most informative) cost partitioning can be found by solving the linear program defined by Equation 4.2. Unfortunately, although linear programming is a polynomial problem, it is far from being an easy one. In practice, solving an LP at *every* evaluated state is very expensive. However, Equation 4.2 does not force us to find an optimal cost partitioning, but rather allows us to choose *any* cost partitioning which obeys the constraints.

One possible ad-hoc cost-partitioning scheme is uniform cost partitioning, which divides the cost of each action equally between all the (relevant)

landmarks it achieves (Karpas and Domshlak, 2009). This cost partitioning is defined by

$$cost(\phi) := \min_{a \in ach(\phi|s,\pi)} \frac{\mathcal{C}(a)}{|L(a \mid s, \pi)|}$$

The following example demonstrates that the uniform cost partitioning may indeed be suboptimal. Consider a planning task with landmarks $\{p_1, p_2, \ldots p_n, q\}$. The possible actions in this task are $a_1 \ldots a_n$, with action a_i achieving p_i and q. Then under uniform cost partitioning, $cost(p_1) = cost(p_2) = \ldots = cost(p_n) = cost(q) = 0.5$, since each action achieves two landmarks, and thus assigns a cost of 0.5 to each of them. This yields a total heuristic estimate of $(n+1)/2$. However, the optimal cost partitioning would assign $cost(p_1) = cost(p_2) = \ldots = cost(p_n) = 1$, $cost(q) = 0$, yielding a total heuristic estimate of n, which is clearly much better.

Fortunately, with a little more inference, it is possible to come up with a cost-partitioning scheme which is almost as fast as uniform cost partitioning, but can be much more informative. This cost-partitioning scheme, which we call the enhanced uniform cost-partitioning scheme, is based on action landmarks. Originally, action landmarks were discovered in a preprocessing phase (Karpas and Domshlak, 2009). However, here we present a method for discovering action landmarks on the fly (Keyder et al., 2010), thus reducing preprocessing time and memory overhead for keeping track of action landmarks, and increasing informativeness (as it is possible to discover that an action landmark that was already used is still needed).

The enhanced uniform cost-partitioning scheme consists of three parts. First, action landmarks are identified during search. An action a is identified as an action landmark after reaching state s via path π, if there exists some landmark $\phi \in LM(s, \pi)$, such that a is the only relevant achiever of ϕ — that is, $ach(\phi \mid s, \pi) = \{a\}$. Second, for any such action landmark discovered, the cost partitioning assigns the full cost of a to ϕ. This can only improve over the uniform cost partitioning, since ϕ must be achieved by a, even though it is possible that a achieves some other landmarks. Finally, uniform cost partitioning is performed over the remaining landmarks and actions.

The benefits of the enhanced uniform cost-partitioning scheme can be seen by considering the previous example. Since $p_1 \ldots p_n$ each have a single achiever ($a_1 \ldots a_n$, respectively), the enhanced uniform cost partitioning scheme would identify $a_1 \ldots a_n$ as action landmarks, and assign their full costs to $p_1 \ldots p_n$, and a cost of 0 to q, thus yielding the optimal cost partitioning (for this example).

4.3 Search with Landmark-Based Heuristics

Regardless of the cost-partitioning scheme used, $LM(s, \pi)$—the set of landmarks that need to be achieved after reaching state s via path π—depends not on the state s, but rather on the path π. Therefore, any heuristic which is based on $LM(s, \pi)$ must be path dependent. One option to get rid of this path dependence is to discover the set of landmarks that need to be achieved from each state being evaluated. However, this would be prohibitively expensive. Instead, we direct our attention to how one can best exploit such heuristics.

4.3.1 Multi-path Dependence

Let us now consider what happens when a state s is reached by two different paths π_1 and π_2. If π_1 and π_2 achieve the same landmarks, then $LM(s, \pi_1) = LM(s, \pi_2)$, and the difference is irrelevant to landmark-based heuristics. However, the case where π_1 and π_2 achieve different landmarks is interesting. Without loss of generality assume that π_1 achieves some landmark ϕ which π_2 does not achieve. Let ρ be some path from s to a goal state. Since ϕ is a landmark, it must be achieved along any plan, including the plan $\pi_2 \cdot \rho$. However, π_2 does not achieve ϕ, and so ρ must achieve ϕ. In our case, where $LM(s, \pi) := \big(LM \setminus Accepted(s, \pi)\big) \cup ReqAgain(s, \pi)$, the above observation can be formulated generally in the following thoerem:

Theorem 4 *Let $\mathcal{P} = \{\pi_1 \dots \pi_n\}$ be a set of paths, all leading from the initial state to some state s. Then any path from s to a goal state must achieve*

$$LM(s, \mathcal{P}) := \big(LM \setminus Accepted(s, \mathcal{P})\big) \cup ReqAgain(s, \mathcal{P}),$$

where

$$Accepted(s, \mathcal{P}) := \bigcap_{\pi \in \mathcal{P}} Accepted(s, \pi)$$

and $ReqAgain(s, \mathcal{P}) \subseteq Accepted(s, \mathcal{P})$ is computed using the same rules as before.

Proof: Assume by contradiction that there exists a path ρ from s to a goal state that does not achieve some landmark $\phi \in LM(s, \mathcal{P})$. Then either $\phi \in LM \setminus Accepted(s, \mathcal{P})$ or $\phi \in ReqAgain(s, \mathcal{P})$. If $\phi \in LM \setminus Accepted(s, \mathcal{P})$ then there must exist some path $\pi_i \in \mathcal{P}$ which does not achieve ϕ (because otherwise, all paths in $\mathcal{P}$ would achieve ϕ, and we would have $\phi \in Accepted(s, \mathcal{P}) = \bigcap_{\pi \in \mathcal{P}} Accepted(s, \pi)$). Therefore $\pi_i \cdot \rho$ achieves ϕ, but π_i

does not achieve ϕ and ρ does not achieve ϕ — a contradiction. Therefore it must be the case that $\phi \in \mathit{ReqAgain}(s, \mathcal{P}) \subseteq \mathit{Accepted}(s, \mathcal{P})$. But ϕ is required again, which means that it does not hold in s, and is either a top-level goal or there exists some other landmark $\phi' \notin \mathit{Accepted}(s, \mathcal{P})$ such that $\phi \ll_{\text{gn}} \phi'$. If ϕ is a top-level goal and does not hold in s, then clearly ρ must achieve ϕ — a contradiction. So there exists $\phi' \notin \mathit{Accepted}(s, \mathcal{P})$ such that $\phi \ll_{\text{gn}} \phi'$. Then, as before, there must exist some path $\pi_j \in \mathcal{P}$ which does not achieve ϕ'. But then $\pi_j \cdot \rho$ must achieve ρ', and since $\phi \ll_{\text{gn}} \phi'$, ϕ must hold in the state prior to the one where ϕ' is first achieved. Since π_j does not achieve ϕ', this must occur somewhere in ρ, which means that ϕ must be achieved again somewhere along ρ — a contradiction.

■

Theorem 4 implies that the heuristic estimate of state s can be improved by using information from multiple paths which reach s. Thus, it is possible to use the information gathered by the search about the structure state-space — $\mathcal{G}_\omega|s$, to derive a more accurate landmark based heuristic, and landmarks can be *multi-path dependent* (Karpas and Domshlak, 2009). It is worth noting that exploiting multi-path dependence can make a large difference in the heuristic values that are computed. For example, consider a planning task with landmarks $p_1 \dots p_n$, where all actions are unit-cost. Assume s is a state which can be reached via n different paths, $\pi_1 \dots \pi_n$, where π_i achieves all the landmarks except p_i. Then, under any cost partitioning scheme, $h_{LA}(\pi_1) = \dots = h_{LA}(\pi_n) = 1$, since according to each of these paths, there is only one more landmark that needs to be achieved. However, combining all of these paths yields the multi-path dependent heuristic estimate $h_{LA}(\{\pi_1, \dots, \pi_n\}) = n$, since the intersection of the achieved landmarks along these paths is empty.

Recall that MPD-A* (Algorithm 2) is adapted to exploit multi-path dependent heuristics. As mentioned in the description of MPD-A*, we do not need to explicitly keep track of the search history. Instead, as in the implementation in LAMA, the only information we store for each state is the set of accepted landmarks $\mathit{Accepted}(s) := \mathit{Accepted}(s, \mathcal{P})$, where $\mathcal{P}$ consists of all known paths to s. When expanding s, for any applicable action a leading from s to s', we simply propagate the accepted landmarks from s to s' and update them according to the landmarks achieved by a — $\mathit{Accepted}(s') := \mathit{Accepted}(s) \cup L(a|s')$, where $L(a|s')$ are the landmarks achieved by a. Since the set intersection operator is monotonic, when a new path $\pi \notin \mathcal{P}$ to a *known* state s is found, we simply perform the update by intersecting the currently stored accepted landmarks $\mathit{Accepted}(s)$ with the

landmarks that are accepted by the new path.

4.3.2 Consistency of Landmark Heuristics

Before we address the question of whether landmark based heuristics are consistent (according to Definition 7), we will show that, using the optimal cost partitioning scheme, landmark heuristics are monotone along single paths. The exact formulation of this appears in the next theorem:

Theorem 5 *Let s be a state reached via path π, and let s' be some successor of s that is reached by applying action a in s. Let h_{LA} be an admissible landmark heuristic as defined in Section 4.2 using optimal cost partitioning. Then it holds that $h_{LA}(\pi) \leq \mathcal{C}(a) + h_{LA}(\pi \cdot \langle a \rangle)$.*

Proof: We will prove this theorem by the following series of inequalities:

$$\begin{aligned}
h_{LA}(\pi) = & \sum_{\phi \in LM(s,\pi)} cost_s(\phi) = \\
& \sum_{\phi \in LM(s,\pi) \setminus L(a|s,\pi)} cost_s(\phi) + \sum_{\phi \in L(a|s,\pi)} cost_s(\phi) \leq \\
& \sum_{\phi \in LM(s,\pi) \setminus L(a|s,\pi)} cost_s(\phi) + \mathcal{C}(a) \leq \\
& \sum_{\phi \in LM(s',\pi \cdot a)} cost_s(\phi) + \mathcal{C}(a) \leq \\
& \sum_{\phi \in LM(s',\pi \cdot a)} cost_{s'}(\phi) + \mathcal{C}(a) = \\
& h_{LA}(\pi \cdot a) + \mathcal{C}(a)
\end{aligned}$$

We begin proving this series of inequalities by noting that, by definition:

$$h_{LA}(\pi) = \sum_{\phi \in LM(s,\pi)} cost_s(\phi)$$

Where $cost_s(\phi)$ is the cost assigned to ϕ under an optimal cost partitioning for s. Clearly $LM(s,\pi) = (LM(s,\pi) \setminus L(a \mid s,\pi)) \cup L(a \mid s,\pi)$, and so:

$$\sum_{\phi \in LM(s,\pi)} cost_s(\phi) = \sum_{\phi \in LM(s,\pi) \setminus L(a|s,\pi)} cost_s(\phi) + \sum_{\phi \in L(a|s,\pi)} cost_s(\phi)$$

By the constraints of Equation 4.2, $\sum_{\phi \in L(a|s,\pi)} cost_s(\phi) \leq \mathcal{C}(a)$, so we have:

$$\sum_{\phi \in LM(s,\pi) \setminus L(a|s,\pi)} cost_s(\phi) + \sum_{\phi \in L(a|s,\pi)} cost_s(\phi) \leq \sum_{\phi \in LM(s,\pi) \setminus L(a|s,\pi)} cost_s(\phi) + \mathcal{C}(a)$$

State s' is obtained by applying action a at state s. Action a achieves landmarks $L(a \mid s, \pi)$, so $Accepted(s', \pi \cdot a) = Accepted(s, \pi) \cup L(a \mid s, \pi)$. Since applying a might cause some more landmarks to become required again, we have: $LM(s, \pi) \setminus L(a \mid s, \pi) \subseteq LM(s', \pi \cdot a)$, and so

$$\sum_{\phi \in LM(s,\pi) \setminus L(a|s,\pi)} cost_s(\phi) + \mathcal{C}(a) \leq \sum_{\phi \in LM(s',\pi \cdot a)} cost_s(\phi) + \mathcal{C}(a)$$

Since an optimal cost partitioning for s can not yield a higher heuristic estimate for s' than an optimal cost partitioning for s', we have:

$$\sum_{\phi \in LM(s',\pi \cdot a)} cost_s(\phi) + \mathcal{C}(a) \leq \sum_{\phi \in LM(s',\pi \cdot a)} cost_{s'}(\phi) + \mathcal{C}(a) = h_{LA}(\pi \cdot a) + \mathcal{C}(a)$$

■

Theorem 5 shows that using optimal cost partitioning guarantees that estimates along one path are consistent. However, this is not guaranteed when using non-optimal cost partitioning. In fact, with non-optimal cost partitioning adding more landmarks can decrease the heuristic value of a state. Consider a planning task with propositions $P = \{p_1 \dots p_n, q\}$, initial state $I = \{q\}$, goal $G = \{p_1 \dots p_n, q\}$, and actions $a_1 \dots a_n, a_q$, with a_i achieving p_i and q $(1 \leq i \leq n)$ and a_q deleting q, with all actions being unit-cost. The landmarks are $p_1 \dots p_n$ and q. Consider state s where q holds. Then the landmarks that need to be achieved are $p_1 \dots p_n$, and the uniform cost partitioning would assign 1 to each of these, for a total estimate of $h_{LA}(s) = n$. Now consider the state s', which is the result of applying a_q from s. q is identified as required again because it is a top-level goal. The landmarks that need to be achieved are $p_1 \dots p_n$ and q, and as seen in Section 4.2.2, this results in an estimate of $h_{LA}(s') = (n+1)/2$ under uniform cost partitioning. For a large enough n it then holds that $h_{LA}(s) > 1 + h_{LA}(s')$.

Furthermore, Theorem 5 can easily be generalized to the case where we calculate the heuristic value of s based on a set of paths $\mathcal{P}$ rather than a

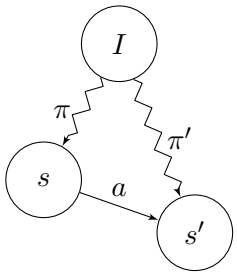


Figure 4.1: Illustration of state space for h_{LA} example

single path π. Then, the requirement to ensure monotonicity is that the set of paths used to calculate the heuristic estimate of s' must be the set of paths used to calculate the heuristic estimate of s, with each path extended by a. However, when different paths could be considered for different states, as is the case when s' can also be reached via a path through a parent state $s'' \neq s$, the heuristic values will not necessarily be consistent.

The state space of such an example is illustrated in Figure 4.1. There are two paths leading to state s': $\pi \cdot \langle a \rangle$ and π'. If $h_{LA}(\pi) = h_{LA}(\pi \cdot \langle a \rangle)$ and $h_{LA}(\pi \cdot \langle a \rangle) - h_{LA}(\pi') > \mathcal{C}(a)$, then the heuristic will not be consistent between s and s'. To see this, note that

$$h_{LA}(\pi) = h_{LA}(\pi \cdot \langle a \rangle) > h_{LA}(\pi') + \mathcal{C}(a).$$

A detailed example demonstrating this behavior, as well as the fact that A^* and MPD-A* will need to reopen some nodes in this problem, is given in Appendix A.

4.4 Existential Optimal Landmarks

We now turn our attention to a different type of landmark — *existential optimal landmarks*, or ∃-opt landmarks for short. While a "regular" landmark must be achieved at some point along *every* plan, an existential optimal landmark might only be achieved along *some* optimal plan. Using these ∃-opt landmarks, we show that the cost-partitioning schemes described previously result in a globally path-admissible heuristic, which can still be used to find an optimal solution.

In this section, we introduce a concrete inference technique that yields such ∃-opt landmarks. Our technique reasons about candidate plan prefixes π generated by the search process, utilizing the well-known notion

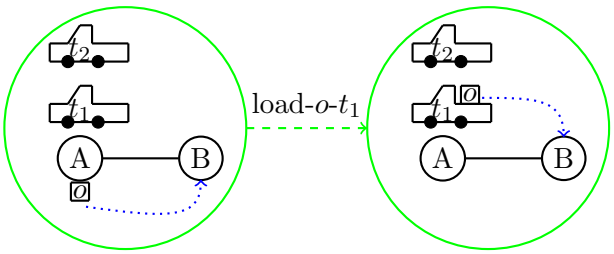


Figure 4.2: Example logistics task

of causal links (Tate, 1977). Causal links are widely exploited in partial-order planning (Penberthy and Weld, 1992; Mcallester and Rosenblitt, 1991), constraint-based planning (Vidal and Geffner, 2006), and recently also in satisficing state space search (Lipovetzky and Geffner, 2011). Our use of causal links here is novel: we use them to infer constraints that must be satisfied by an optimal plan having π as its prefix, and then use these constraints to enhance the heuristic evaluation of the end-state of π.

The technique is based on a simple observation that, for each action along every optimal plan for the problem, if there is no justification for applying that action, then it can be removed, yielding a shorter plan. Consider a simple logistics problem, depicted in Figure 4.2, with two locations A and B, two trucks t_1 and t_2, and a single package. In the initial state both trucks and the package are at location A, and the goal is to have the package at B. Clearly, any solution must load the package into one of the trucks. Now, suppose we have already loaded the package onto truck t_1. While it is still possible to unload the package from t_1, load it onto t_2, and use t_2 to deliver the package to B, any *optimal* solution from the state in question will exploit the fact that some effort has already been put into loading the package onto t_1, and will use t_1 to deliver the package. This is precisely the type of assertions our inference technique tries to deduce.

4.4.1 Intended Effects

Before we describe our inference technique in detail, we must first define the notion of *minimal preconditions* after following path π. Intuitively, X is a minimal precondition after following I-path π, if π achieves X and X is "needed" to continue π to an optimal plan. However, the formal definition distinguishes between the minimal preconditions of an I-path π in the

context of different sets of optimal plans:

Definition 15 (Minimal Precondition)
Let $\Pi = \langle P, I, G, A, \mathcal{C} \rangle$ *be a planning task, and* OPT *be a set of optimal plans for* Π*. Given an* I*-path* $\pi = \langle a_0, a_1, \ldots a_n \rangle$*, a set of propositions* $X \subseteq I[\![\pi]\!]$ *is an* OPT-minimal precondition after π *iff there exists an* X*-plan* π' *such that* $\pi \cdot \pi' \in$ OPT *and for every* $X' \subset X$*,* π' *is not applicable in* X'*.*

In other words, X is an OPT-minimal precondition after π if π achieves X, and it is possible to continue after π into some plan in OPT, using only facts in X. Furthermore, we require that X is minimal with regards to set inclusion. While Definition 15 is fairly intuitive, for the purposes of explaining our inference technique we use the following equivalent definition, which is a bit more involved:

Definition 16 (Intended Effect)
Let $\Pi = \langle P, I, G, A, \mathcal{C} \rangle$ *be a planning task, and* OPT *be a set of optimal plans for* Π*. Given an* I*-path* $\pi = \langle a_0, a_1, \ldots a_n \rangle$*, a set of propositions* $X \subseteq I[\![\pi]\!]$ *is an* OPT-intended effect of π *iff there exists an* $I[\![\pi]\!]$*-plan* π' *such that* $\pi \cdot \pi' \in$ OPT *and* $X = \{p \mid \langle a_i, p, a_j \rangle$ *is a causal link in* $\pi \cdot \pi', a_i \in \pi, a_j \in \pi'\}$

In other words, X is an OPT-intended effect of π if π achieves X, and it is possible to use π' after π so that the full plan $\pi \cdot \pi' \in$ OPT, and π' consumes *exactly* X, that is, $p \in X$ iff there is a causal link $\langle a_i, p, a_j \rangle$ in $\pi \cdot \pi'$, with $a_i \in \pi$ and $a_j \in \pi'$.

The basic observation underlying the notion of intended effect is very simple, if not to say trivial: every action along an optimal plan should be there for a reason, that is, there should be some use of at least one of the effects of each plan's action. The following theorem shows that Definitions 15 and 16 are equivalent, under a simple compilation of the planning task which adds a dummy START action:

Theorem 6 (Definition Equivalence)
Let $\Pi = \langle P, I, G, A, \mathcal{C} \rangle$ *be a planning task with* $I = \emptyset$*, and a unique* START *action, let* OPT *be a set of optimal plans for* Π*, let* $\pi = \langle a_0, a_1, \ldots a_n \rangle$ *be some* I*-path, and let* $X \subseteq I[\![\pi]\!]$*. Then* X *is an* OPT*-minimal precondition after* π *iff* X *is an* OPT*-intended effect of* π*.*

Proof: Let X be a OPT-minimal precondition after π. Then there exists some path π', such that $\pi \cdot \pi' \in$ OPT, and π' is applicable in X, but not in any proper subset of X.

To show that X is an OPT-intended effect of π, we must show that $p \in X$ iff there exists some causal link $\langle a_i, p, a_j \rangle$ in $\pi \cdot \pi'$, with $a_i \in \pi$ and $a_j \in \pi'$. Denote the achiever of $p \in I[\![\pi]\!]$ by $ach(p) := \max_{\{i|0 \leq i \leq n, p \in \mathsf{add}(a_i)\}} i$. Every proposition must have an achiever, because $I = \emptyset$.

If $p \in X$, then it must be a precondition of some action in π', because otherwise π' would be applicable in $X \setminus \{p\}$. Denote the first action in π' which has p as a precondition by $cons(p)$. Then clearly $\langle ach(p), p, cons(p) \rangle$ is a causal link as required.

If $p \notin X$, then either (a) there is no action in π' which has p as a precondition, or (b) there is some action $a_j \in \pi'$ which has p as a precondition, and there is some action $a_i \in \pi'$ with $i < j$ which achieves p. Otherwise, π' would not be an X-plan. In case (a), there is clearly no causal link on p with a consumer in π', as there is no action in π' which requires p. In case (b) denote the first action in π' which requires p by a_j, and denote by $a_i \in \pi'$ the latest action to achieve p before a_j. $\langle a_i, p, a_j \rangle$ is a causal link, but the producer is in π', and so there is no causal link with the producer in π as required. Thus, $p \in X$ iff there exists some causal link $\langle a_i, p, a_j \rangle$ in $\pi \cdot \pi'$, with $a_i \in \pi$ and $a_j \in \pi'$, and X is an OPT-intended effect of π.

Now assume X is an OPT-intended effect of π. Then there exists some path π', such that $p \in X$ iff there exists some causal link $\langle a_i, p, a_j \rangle$ in $\pi \cdot \pi'$, with $a_i \in \pi$ and $a_j \in \pi'$. We will show that X is an OPT-minimal precondition after π (that is, that π' is applicable in X, but not in any proper subset of X).

Assume to the contrary that π' is not applicable in X. Denote by a_j the first action in π' which is not applicable, and denote by $p \in \mathsf{pre}(a_j)$ some proposition that does not hold before applying a_j (after following π' until a_j from state X). If $p \in X$, then there is some causal link $\langle a_i, p, a_j \rangle$ in $\pi \cdot \pi'$, with $a_i \in \pi$ and $a_j \in \pi'$. But then p could not have been deleted before a_j, and $p \in X$, which means that p must hold before applying a_j — a contradiction. If $p \notin X$, then there is no causal link between π and π' on p. Therefore, when applying π' in $I[\![\pi]\!]$, p must be achieved by some action in π'. But then, when applying π' in X, a_j should be applicable — a contradiction.

Therefore, π' is applicable in X. We must now show that there is no $X' \subset X$, such that π' is applicable in X'. Assume to the contrary that there exists such X', and let $p \in X \setminus X'$. $p \in X$, so there must exist some causal link $\langle a_i, p, a_j \rangle$ in $\pi \cdot \pi'$, with $a_i \in \pi$ and $a_j \in \pi'$. π' is applicable in X', but $p \notin X'$, implying that some action in π' achieved p for a_j. But $\langle a_i, p, a_j \rangle$ is a causal link in $\pi \cdot \pi'$, with $a_i \in \pi$, implying that there is no action that achieves p before a_j in π' — a contradiction. ∎

Having seen the OPT-intended effects and OPT-minimal preconditions are the same, we denote the set of all OPT-intended effects of an I-path π by $IE(\pi \mid \mathrm{OPT})$; when OPT is the set of *all* optimal plans for Π, then $IE(\pi \mid \mathrm{OPT})$ is simply called "intended effects" and is denoted by $IE(\pi)$. Note that if π is not a cheapest path from I to $I[\![\pi]\!]$ then $IE(\pi \mid \mathrm{OPT}) = \emptyset$ for all optimal plan sets OPT.

If provided to us, the intended effects of π can reveal valuable information about what any continuation of π must do. For example, if for some proposition p we have $p \in X$ for all intended effects $X \in IE(\pi)$, then clearly any optimal continuation of π must contain some action consuming p. This example suggests that intended effects of π can be used either for deriving a heuristic estimate of $I[\![\pi]\!]$, or for enhancing such an existing estimate. We now suggest one such framework for exploiting intended effects. It is based on what we call *existential optimal-plan landmarks*, or $\exists$*-opt landmarks*, for short.

First, interpreting proposition subsets $X \subseteq P$ as valuations of P, assume that a set of intended effects $IE(\pi \mid \mathrm{OPT})$ is given to us as a propositional logic formula ϕ such that that $X \in IE(\pi \mid \mathrm{OPT}) \Leftrightarrow X \models \phi$. By $models(\phi)$ we denote the set of ϕ's models, that is, $models(\phi) = \{X \subseteq P \mid X \models \phi\}$. For an I-path π, let us also treat any continuation π' of π as a valuation of P, assigning $TRUE$ to the propositions produced by π *and* consumed by π', and $FALSE$ to all other propositions. This way, the semantics of statements "π' satisfies ϕ", $\pi' \models \phi$, is well defined.

Theorem 7 *Let* OPT *be a set of optimal plans for a planning task* Π*,* π *be an* I*-path, and* ϕ *be a propositional logic formula describing* $IE(\pi \mid \mathrm{OPT})$*. Then, for any* $I[\![\pi]\!]$*-plan* π'*,* $\pi \cdot \pi' \in \mathrm{OPT}$ *implies* $\pi' \models \phi$*.*

Theorem 7, proof of which is immediate from Definition 16, establishes our interpretation of the formula ϕ as an $\exists$-opt landmark: ϕ is not a landmark in the standard sense of this term (Hoffmann et al., 2004), as not every plan, and not even every optimal plan, must satisfy ϕ. However, *some optimal* plan starting with π must satisfy ϕ after π.

In line with the recent work on regular landmarks, henceforth we assume that ϕ is given in CNF. The CNF representation of ϕ is advantageous mainly in that it has a natural interpretation as a set of disjunctive fact landmarks, where each clause describes one such landmark. Note that unlike the regular landmarks, where a fact landmark stands for a disjunctive action landmark composed of its *achievers*, in our $\exists$-opt landmarks a fact stands for a disjunctive action landmark composed of its *consumers*. How-

ever, it is possible, for instance, to combine the information captured by the $\exists$-opt landmark(s) ϕ and the information captured by the regular landmarks of the h_{LA} heuristic, by performing an action cost partition over the union of their landmarks (Karpas and Domshlak, 2009). When cost partitioning is optimized via, e.g., the linear programming technique (Karpas and Domshlak, 2009; Katz and Domshlak, 2010a), the resulting estimate is guaranteed to dominate h_{LA}. In fact, if we could find just a single OPT-intended effect $X \in IE(\pi \mid \text{OPT})$, we could then use X as a *regular* landmark, pruning some parts of the search space without sacrificing the optimality: Since we know there must exist some continuation π' with $\pi \cdot \pi' \in \text{OPT}$, X by itself constitutes an $\exists$-opt landmark.

So far, we have outlined the promise of $\exists$-opt landmarks induced by intended effects, yet that promise is, of course, only potential since the intended effects $IE(\pi \mid \text{OPT})$ were assumed to be somehow provided to us. It is hardly surprising, however, that finding just a single intended effect of an action sequence is as hard as STRIPS planning itself.

Theorem 8 *Let INTENDED be the following decision problem: Given a planning task $\Pi = \langle P, I, G, A, \mathcal{C} \rangle$, an I-path π, and a set of propositions $X \subseteq P$, is $X \in IE(\pi)$?*

Deciding INTENDED is $PSPACE - hard$.

Proof: The proof is by reduction from the complement of PLANSAT — the problem of deciding whether a given planning task is solvable. For STRIPS, PLANSAT is known to be $PSPACE - hard$ even when all actions are unit cost (Bylander, 1994), and since PSPACE = CO-PSPACE, so is its complement.

Given a planning task $\Pi = \langle P, I, G, A, \mathcal{C} \rangle$ with unit cost actions and $|P| = n$, we construct a new planning task $\Pi' = \langle P', I', G', A', \mathcal{C}' \rangle$ as follows:

- $P' := P \cup \{d_i \mid 0 \leq i \leq n+1\}$;
- $I' := I$;
- $G' := G \vee (d_0 \wedge d_1 \wedge \ldots \wedge d_{n+1})$.
- $A' := A \cup \{\text{inc}(i) \mid 0 \leq i \leq n+1\}$, where $\text{inc}(i) = \langle \{d_j \mid j < i\}, \{d_i\}, \{d_j \mid j < i\} \rangle$; and
- $\mathcal{C}'$ assigns costs of 1 to all actions in A;

The goal G' is disjunctive, but this disjunction can be straightforwardly compiled away.

Note that Π' is always solvable because there will always be a solution of cost $2^{n+2} - 1$, using the inc operators to increment a binary counter composed of $d_0, \ldots, d_{n+1}$. Now, if the original task Π is solvable, then it has a solution of cost at most $2^n - 1$. Therefore, the inc operators and the d_i propositions are part of an optimal solution iff Π is not solvable, and thus $\{d_0\}$ is an optimal intended effect in Π' after applying inc(0) iff Π is not solvable. ∎

Although Theorem 8 shows that computing $IE(\pi \mid \mathrm{OPT})$ precisely is not feasible, the promise of ∃-opt landmarks still remains: we can approximate $IE(\pi \mid \mathrm{OPT})$ while still guaranteeing optimality, and thus maintain the correctness of the reasoning. In particular, below we show that any superset of $IE(\pi \mid \mathrm{OPT})$ induces *possible intended effects* and provides such a "safe" approximation.

Theorem 9 *Let* OPT *be a set of optimal plans for a planning task* Π*,* π *be an I-path,* $PIE(\pi \mid \mathrm{OPT}) \supseteq IE(\pi \mid \mathrm{OPT})$ *be a set of possible* OPT*-intended effects of* π*, and* ϕ *be a logical formula describing* $PIE(\pi \mid \mathrm{OPT})$*. Then, for any* $I[\![\pi]\!]$*-plan* π'*,* $\pi \cdot \pi' \in \mathrm{OPT}$ *implies* $\pi' \models \phi$*.*

Proof: Let π' be an $I[\![\pi]\!]$-plan such that $\pi \cdot \pi' \in \mathrm{OPT}$, and let X be the set of all propositions produced by π and consumed by π'. From Definition 16, $X \in IE(\pi \mid \mathrm{OPT})$, and since $IE(\pi \mid \mathrm{OPT}) \subseteq PIE(\pi \mid \mathrm{OPT})$, $X \in PIE(\pi \mid \mathrm{OPT})$. Since ϕ describes $PIE(\pi \mid \mathrm{OPT})$, it holds that $X \models \phi$. ∎

We now proceed with describing a concrete proposal for finding and utilizing useful *PIE* approximations of this type in the context of OPT containing either all optimal plans or just one optimal plan.

4.4.2 Approximating Intended Effects

One easy way of obtaining a set $PIE(\pi)$ such that $IE(\pi) \subseteq PIE(\pi)$ is to take $PIE(\pi) = 2^P$. Needless to say, however, it provides us with no useful information whatsoever. A slightly tighter approximation of $IE(\pi)$ would be $PIE(\pi) = 2^{I[\![\pi]\!]}$. Clearly, no continuation of π can consume anything that π achieved but does not hold in the state reached by π. However, this approximation of $IE(\pi)$ still does not provide us with any useful information.

We begin by showing that it is possible to obtain a much tighter approximation of $IE(\pi \mid \mathrm{OPT})$ for OPT consisting of a single optimal plan ρ, and that this approximation can provide us with useful information about the OPT-continuations of π. Obviously, the plan ρ will not actually be known

to us; or otherwise there would be no point in planning in the first place. In itself, however, that will not be an obstacle.

This approximation is based on exploiting a library $\mathcal{L}$ of I-paths; later we discuss how such a library can be obtained automatically, but for now we assume we are simply provided with one. Given a planning task Π, let $\prec$ be a lexicographic order on its action sequences, based on an arbitrary total order of the actions. Let ρ be the optimal plan for Π that is minimal (lowest) with respect to $\prec$. That is the plan we focus on, and we can now describe our approximation of $IE(\pi \mid \{\rho\})$. For that, we define an additional ordering $\lhd$ on action sequences:

$$\pi' \lhd \pi \Leftrightarrow \mathcal{C}(\pi') < \mathcal{C}(\pi) \vee (\mathcal{C}(\pi') = \mathcal{C}(\pi) \wedge \pi' \prec \pi).$$

Note that $\pi' \lhd \pi$ implies $\mathcal{C}(\pi') \leq \mathcal{C}(\pi)$. Using $\lhd$, we approximate $IE(\pi \mid \{\rho\})$ with

$$PIE_{\mathcal{L}}(\pi \mid \{\rho\}) = \{X \subseteq I[\![\pi]\!] \mid \nexists \pi' \in \mathcal{L} : \pi' \lhd \pi, X \subseteq I[\![\pi']\!]\}.$$

In other words, for any $X \subseteq I[\![\pi]\!]$, $\mathcal{L}$ *proves* that X is *not* an intended effect of π if it can offer a cheaper way of achieving X from I, or if it can offer a way of achieving X from I at the same cost, but that alternative way is "preferred" to π with respect to $\prec$.

Theorem 10 *Let $\prec$ be a lexicographic order on action sequences, and let ρ be an optimal solution of Π that is minimal in $\prec$. For any I-path π, it holds that $IE(\pi \mid \{\rho\}) \subseteq PIE_{\mathcal{L}}(\pi \mid \{\rho\})$.*

Proof: Assume to the contrary that there exists some $X \in IE(\pi \mid \{\rho\}) \setminus PIE_{\mathcal{L}}(\pi \mid \{\rho\})$. Since $X \in IE(\pi \mid \{\rho\})$, there exists some path π' such that $\pi \cdot \pi' = \rho$, and π' consumes all propositions in X. $X \notin PIE_{\mathcal{L}}(\pi \mid \{\rho\})$, so from the definition of $PIE_{\mathcal{L}}(\pi \mid \{\rho\})$ there exists some I-path $\pi'' \in \mathcal{L}$ such that π'' is applicable at I, $\pi'' \lhd \pi$, and $X \subseteq I[\![\pi'']\!]$.

π' is applicable at $I[\![\pi'']\!]$, since π' consumes exactly the propositions in X, and $X \subseteq I[\![\pi'']\!]$. $\pi \cdot \pi' = \rho$ is a valid plan, so the last action in π' must be the END action, which implies that $\pi'' \cdot \pi'$ is a valid plan.

$\pi'' \lhd \pi$, and so one of (I-II) below must be true:

I $\mathcal{C}(\pi'') < \mathcal{C}(\pi)$.

But then, $\mathcal{C}(\pi'' \cdot \pi') < \mathcal{C}(\pi \cdot \pi')$, contradicting the optimality of $\pi \cdot \pi' = \rho$.

II $\mathcal{C}(\pi'') = \mathcal{C}(\pi)$ and $\pi'' \prec \pi$.

But then $\mathcal{C}(\pi'' \cdot \pi') = \mathcal{C}(\pi \cdot \pi')$, and thus $\pi'' \cdot \pi'$ is an optimal plan. Since $\prec$ is a lexicographic order, $\pi'' \cdot \pi' \prec \pi \cdot \pi'$, contradicting the minimality of $\pi \cdot \pi' = \rho$ in $\prec$.

We have seen that either case of $\pi'' \triangleleft \pi$ leads to a contradiction, thus proving the theorem. ∎

We note that, very similarly, one can obtain an approximation of the intended effects $IE(\pi)$ of π with respect to *all* optimal plans, with no need for the lexicographic order $\prec$ on the action sequences. Specifically, one can use

$$PIE_{\mathcal{L}}(\pi) = \{X \subseteq I[\![\pi]\!] \mid \nexists \pi' \in \mathcal{L} : \mathcal{C}(\pi') < \mathcal{C}(\pi), X \subseteq I[\![\pi']\!]\},$$

and the proof that $IE(\pi) \subseteq PIE_{\mathcal{L}}(\pi)$ is very similar to the proof of Theorem 10.

4.4.3 From $PIE_{\mathcal{L}}(\pi \mid \{\rho\})$ to Existential Optimal Landmarks

As mentioned, in order to use $PIE_{\mathcal{L}}(\pi \mid \{\rho\})$ as a set of disjunctive fact landmarks, we have to derive a CNF formula that compactly represents it. Recall that $PIE_{\mathcal{L}}(\pi \mid \{\rho\})$ consists of all sets of propositions in $I[\![\pi]\!]$ for which there is no "shortcut" in $\mathcal{L}$. Let π be an I-path, and let $\pi' \in \mathcal{L}$ be an I-path in the library such that $\pi' \triangleleft \pi$. If π is a prefix of our $\prec$-minimal optimal plan ρ, then there must be some proposition p consumed by the continuation of π along ρ which is achieved by π but not by π', that is, $p \in I[\![\pi]\!] \setminus I[\![\pi']\!]$. In CNF, this information, derived from π on the basis of the library $\mathcal{L}$, is encoded as

$$\phi_{\mathcal{L}}(\pi|\{\rho\}) = \bigwedge_{\pi' \in \mathcal{L} : \pi' \triangleleft \pi} \bigvee_{p \in I[\![\pi]\!] \setminus I[\![\pi']\!]} p.$$

As a special case of $\phi_{\mathcal{L}}(\pi|\{\rho\})$, note that if there exists $\pi' \in \mathcal{L}$ with $\mathcal{C}(\pi') < \mathcal{C}(\pi)$ and $I[\![\pi]\!] \subseteq I[\![\pi']\!]$, then $\phi_{\mathcal{L}}(\pi|\{\rho\})$ contains an empty clause, meaning that there is no optimal continuation of π. This is the case captured by the definition of a dominating action sequence (Nedunuri et al., 2011), and $\phi_{\mathcal{L}}(\pi|\{\rho\})$ generalizes their definition. The following theorem demonstrates that $\phi_{\mathcal{L}}(\pi|\{\rho\})$ describes $PIE_{\mathcal{L}}(\pi \mid \{\rho\})$, and thus by Theorem 10 approximates $IE(\pi \mid \{\rho\})$:

Theorem 11 *For any I-path π, $PIE_{\mathcal{L}}(\pi \mid \{\rho\}) = models\big(\phi_{\mathcal{L}}(\pi|\{\rho\})\big)$.*

Proof: We first show that $PIE_{\mathcal{L}}(\pi \mid \{\rho\}) \subseteq models\big(\phi_{\mathcal{L}}(\pi|\{\rho\})\big)$. Assume to the contrary that there exists $X \in PIE_{\mathcal{L}}(\pi \mid \{\rho\}) \setminus models\big(\phi_{\mathcal{L}}(\pi|\{\rho\})\big)$. $X \notin models\big(\phi_{\mathcal{L}}(\pi|\{\rho\})\big)$, so there exists some clause $c_{\pi'} = \vee_{p \in I[\![\pi]\!] \setminus I[\![\pi']\!]} p$ in $\phi_{\mathcal{L}}(\pi|\{\rho\})$, corresponding to path $\pi' \in \mathcal{L}$, which X does not satisfy. Since the clause $c_{\pi'}$ contains the propositions $I[\![\pi]\!] \setminus I[\![\pi']\!]$, this implies that $X \cap \big(I[\![\pi]\!] \setminus I[\![\pi']\!]\big) = \emptyset$. We know that $X \subseteq I[\![\pi]\!]$, and so we have that $X \subseteq I[\![\pi']\!]$. However, $X \in PIE_{\mathcal{L}}(\pi \mid \{\rho\})$, so there is no "shortcut" in $\mathcal{L}$ for achieving X. Therefore, for any $\pi' \in \mathcal{L}$ such that $\pi' \triangleleft \pi$, we have that $X \not\subseteq I[\![\pi']\!]$—a contradiction.

We now show that $models\big(\phi_{\mathcal{L}}(\pi|\{\rho\})\big) \subseteq PIE_{\mathcal{L}}(\pi \mid \{\rho\})$. Assume to the contrary that there exists $X \in models\big(\phi_{\mathcal{L}}(\pi|\{\rho\})\big) \setminus PIE_{\mathcal{L}}(\pi \mid \{\rho\})$. $X \notin PIE_{\mathcal{L}}(\pi \mid \{\rho\})$, so there exists some $\pi' \in \mathcal{L}$ such that $\pi' \triangleleft \pi$ and $X \subseteq I[\![\pi']\!]$. Let $c_{\pi'} = \vee_{p \in I[\![\pi]\!] \setminus I[\![\pi']\!]} p$ be the clause corresponding to π' in $\phi_{\mathcal{L}}(\pi|\{\rho\})$. We know that $X \subseteq models\big(\phi_{\mathcal{L}}(\pi|\{\rho\})\big)$, thus X must satisfy every clause of $\phi_{\mathcal{L}}(\pi|\{\rho\})$, and specifically, X must satisfy $c_{\pi'}$. However, $X \subseteq I[\![\pi']\!]$, and $c_{\pi'}$ does not contain any of these propositions. Thus, X cannot satisfy $c_{\pi'}$ — a contradiction. ∎

Similarly, we can derive a CNF formula $\phi_{\mathcal{L}}(\pi)$ which describes $PIE(\pi)$ (with a very similar proof of the equivalence):

$$\phi_{\mathcal{L}}(\pi) = \bigwedge_{\pi' \in \mathcal{L}: \mathcal{C}(\pi') < \mathcal{C}(\pi)} \bigvee_{p \in I[\![\pi]\!] \setminus I[\![\pi']\!]} p.$$

4.4.4 Obtaining a Shortcut Library

So far we assumed our "shortcut" library $\mathcal{L}$ is given. We now describe a concrete approach to obtaining it. Importantly, note that $\mathcal{L}$ does not have to be a static list of action sequences, but rather can be generated dynamically for each I-path π constructed by the search procedure. In particular, such a path-specific library can be generated using a set of rules similar to plan rewrite rules by Nedunuri et al. (2011). A plan rewrite rule is a rule of the form $\pi_1 \rightarrow \pi_2$, where π_1 and π_2 are some action sequences and the rule means that *whenever* π_1 is a subsequence of a plan, it can be replaced on that plan with π_2 without violating the plan's validity. We do not require such a strong connection between π_1 and π_2. First, instead of requiring that π_2 be applicable whenever π_1 is applicable, we can simply check whether π_2 is applicable for the current state. Second, we do not require π_2 to achieve everything that π_1 does, since we can also obtain useful information from the set difference of the end states of π_1 and π_2.

Given a path π and a set of plan rewrite rules, we construct a library $\mathcal{L}(\pi)$ specifically for π. To construct $\mathcal{L}(\pi)$, we first need to define a digraph, which we call the *causal structure of* π. The nodes of this graph are the action instances in π, and there is an edge from a_i to a_j when there is a causal link in π with a_i as the provider and a_j as the consumer. Given a plan rewrite rule $\pi_1 \rightarrow \pi_2$ with $\pi_2 \lhd \pi_1$, we can check whether π_1 appears as a chain in the causal structure of π. We can then attempt to replace π_1 with π_2, and check if the resulting action sequence is still applicable in I. Note that π_1 does not have to appear in π as a contiguous subsequence, making this a more general strategy than simply looking for contiguous occurrences of π_1.

Our current implementation is a special case of this scheme that uses plan rewrite rules of the form $\pi \rightarrow \langle\rangle$, that is, the tail of each rule is the empty sequence. We look for two types of chains in the causal structure of π. The first type are isolated chains, that is, chains with no edges going out from any node in the chain to any node outside the chain. This ensures that removing every suffix of the causal chain still results in a valid I-path. As a special case of this, removing the last operator in each causal chain will yield the same landmarks as those from the "unjustified actions" (Karpas and Domshlak, 2011), up to ordering of the action sequence. The second type of chains we look for correspond to "action a supports its inverse action a'", where the notion of action invertibility is adopted from Hoffmann (2002).

To illustrate how this process works, we consider the following example action sequence in a Logistics problem: $\pi = \langle drive(t_1, A, B), drive(t_1, B, C), drive(t_2, A, B), drive(t_1, C, A)\rangle$. The causal structure of this action sequence is shown in Figure 4.3. Clearly, truck t_1 drives in a loop here, without doing anything useful on the way. Thus, the actions $drive(t_1, A, B)$, $drive(t_1, B, C)$, and $drive(t_1, C, A)$ form an isolated causal chain. We can replace this causal chain with the empty sequence, yielding the "shortcut" $\pi' = \langle drive(t_2, A, B)\rangle$, which leads to the same state as π, allowing us to prove that π cannot be the beginning of any optimal solution. Note that while truck t_1 drives in a loop, there is no loop in the state space (which could be detected by the search algorithm), since truck t_2 moved in between the moves of t_1. However, our shortcut library allows us to eliminate non-contiguous subsequences of π, making this stronger than simple duplicate state detection.

In terms of previous uses of causal links, the PROBE planner's (Lipovetzky and Geffner, 2011) causal commitments bear similarity to our use of the causal structure, with one major difference. While PROBE chooses the causal commitments (that is, which proposition each action *will* be used

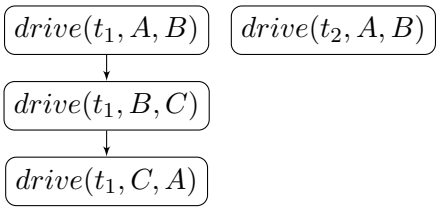


Figure 4.3: Causal structure of example

to achieve) as soon as the action is chosen, we look at all possible causal commitments. This is why PROBE cannot guarantee optimality, while we can.

4.4.5 Search with ∃-opt Landmark Heuristics

For a set of optimal plans OPT, it is fairly easy to see that given a CNF formula $\phi_{\mathcal{L}}(\pi|\text{OPT})$, which describes some sound approximation of $IE(\pi \mid \text{OPT})$, any admissible action cost partition over $\phi_{\mathcal{L}}(\pi|\text{OPT})$ yields an OPT-admissible heuristic. Specifically, any admissible action cost partition over $\phi_{\mathcal{L}}(\pi|\{\rho\})$ yields a $\{\rho\}$-admissible heuristic, and any admissible action cost partition over $\phi_{\mathcal{L}}(\pi)$ yields a path admissible heuristic.

As previously mentioned, *path-A** using a path admissible heuristic is guaranteed to find an optimal solution. Unfortunately, *path-A** is not guaranteed to find an optimal solution with a globally path admissible heuristic, and specifically with a $\{\rho\}$-admissible heuristic.

However, it is possible to make a small modification of *path-A**, that will ensure that an optimal solution is found with a $\{\rho\}$-admissible, for our specific ρ — the $\prec$-minimal lowest optimal plan. The modification is very simple: instead of reopening a state only when a cheaper to it has been found, we also reopen it when a new path of the same cost, but lower according to $\prec$ is found. In terms of pseudo code, we only need to change the condition for reopening a state in line 18 of *path-A** (Algorithm 3), from:

$$\overline{g}(s) + \mathcal{C}(a) < \overline{g}(s')$$

to:

$$\big(trace(s) \cdot \langle a \rangle\big) \lhd trace(s')$$

Using this modification of *path-A** with a heuristic derived from $\phi_{\mathcal{L}}(\pi|\{\rho\})$ is guaranteed to find an optimal solution. To see this, we prove the A^* Lemma for this variant of *path-A**.

Lemma 3
At any time before (the modified) path-A^* terminates, there exists on the open list a state s which lies on ρ (the $\prec$-minimal optimal plan), with $\overline{f}(s) \leq c^*$

Proof: Assume the states which lie along ρ are $\langle I, s_1, \ldots, s, \ldots, s_n \rangle$, and let s be the shallowest state on ρ which is open (there is at least one, because the goal state s_g is only closed when *path-A^** terminates). Since all ancestors of s along ρ are closed, and the new condition for reopening will always reopen prefixes of ρ, it must be that the parent pointers assigned to s and its ancestors are along ρ. Since ρ is optimal, we know that $\overline{g}(s) = g^*(s)$. Using the $\{\rho\}$-admissibility of h, we obtain:

$$\overline{f}(s) = \overline{g}(s) + \overline{h}(s) = g^*(s) + \overline{h}(s) \leq g^*(s) + h^*(s) = c^*$$

■

As is the case with the unmodified *path-A^**, using this lemma with the proof of Theorem 1 also shows that the modified *path-A^** with a $\{\rho\}$-admissible heuristic is guaranteed to find an optimal solution.

4.5 Future Work

We believe this is not the end of the road for ∃-opt landmarks. First, the dynamic shortcut library generation process can be improved by introducing more general forms of plan rewrite rules — not just rules which attempt to delete some action sequences from the current path, but rules which attempt to *replace* some action sequences with other action sequences. There are several possible sources for these rules, including learning them online, during search. This would make ∃-opt landmarks history dependent, as mentioned in Table 3.1.

Additionally, it is quite likely that other methods of deriving ∃-opt landmarks could be found. In fact, the inference technique we present here could be enhanced with additional reasoning, as demonstrated in the following scenario. Assume that action a was applied, and achieved proposition p. Our current inference technique can deduce that at some later point, some action which consumes p must be applied. Still, the question is *when* a consumer of p should be applied. One natural option is to apply it directly after a. However, there are two possible reasons this might not be the best choice: either the consumer requires some other preconditions which have not yet

been achieved, or the consumer threatens another action, which should be applied before the consumer. Incorporating this type of reasoning into our inference technique poses an interesting challenge.

Chapter 5

Machine-Learning Based Heuristics

Another class of non-classical heuristics are those that are based on online learning. The intractability of classical planning has led many researchers to use speedup learning techniques in order to improve the performance of planning systems; for a survey of many of these, see Minton (1994); Zimmerman and Kambhampati (2003); Fern et al. (2011).

In general, speedup learning is concerned with improving the performance of a problem solving system with experience. Speedup learning systems can be divided along several dimensions (Zimmerman and Kambhampati, 2003; Fern, 2010). Arguably the most important dimension is the phase in which learning takes place. An offline, or inter-problem, speedup learner analyzes the performance of the problem solver on different problem instances, and attempts to formulate some rule which would improve performance of the solver on the observed example instances, and would hopefully generalize well to future problem instances. Offline learning has been applied to domain independent planning extensively, with varying degrees of success (Fern et al., 2011). However, one major drawback of offline learning is the need for training examples — in our case, planning tasks from the domains of interest.

The other phase in which learning could take place is online, during problem solving. An *online, or intra-problem, speedup learner* is invoked by the problem solver on a concrete problem instance the solver is working on, and it attempts to learn online, with the objective of improving the performance of the solver on the concrete problem instance in question. In general, online learners are not assumed to be pre-trained on some other, previously seen

problem instances; all the information they can rely on has to be collected during the process of solving the concrete problem instance they were called for. Online learning has been shown to be extremely helpful in propositional satisfiability (SAT) and general constraint satisfaction (CSP) solving, where nogood learning and clause learning are now of the essential components of any state of the art solver (Schiex and Verfaillie, 1993; Marques-Silva and Sakallah, 1996; Bayardo Jr. and Schrag, 1997). Thus, indirectly, SAT- or CSP-based domain-independent planners already benefit from these online learning techniques (Kautz and Selman, 1992; Rintanen et al., 2006).

In this chapter, we present an online learning approach for speeding up optimal heuristic-search planning. Our approach, *selective max*, is based on combining the strengths of several heuristic evaluation functions. At a high level, it can be seen as a hyper-heuristic (Burke et al., 2003) — a heuristic for choosing between other heuristics. Specifically, selective max is based on a seemingly useless observation that, for each state there is one heuristic which is the "best" for that state. Although it is possible to compute several heuristics for each state, and then choose one of them based on the values they provide, computing several heuristic estimates for each state takes a significant amount of time.

Selective max works by predicting for each state which heuristic will yield the "best" heuristic estimate, and computes only that heuristic. As it is not always clear how to decide what the "best" heuristic for each state is, we first analyze an idealized model of a search space and describe how an all-knowing oracle would choose the best heuristic for each state, when the objective is to minimize the overall search time. Using the oracle's decision as a target concept, we then describe an online active learning procedure for that concept, which constitutes the essence of selective max. Finally, it is worth mentioning that the resulting heuristic is history-dependent — the value assigned to some state s can depend on information that was learned from a different state s', which does not lie on any path to s.

5.1 High-Level Overview

In domain-independent planning, many admissible heuristics have been proposed, varying from cheap to compute yet typically not very accurate, to expensive to compute but much more accurate. As the accuracy of heuristic functions varies for different planning tasks, and even for different states of the same task, we can produce a more robust optimal planner by combining several admissible heuristics. The simplest and best-known way for doing

that is using the point-wise maximum of the heuristics in use at each state. Presumably, each heuristic is more accurate, that is, provides higher estimates, in different regions of the search space, and thus point-wise maximum is at least as accurate as each of the individual heuristics.

As mentioned previously, selective max can be seen as a type of hyper-heuristic (Burke et al., 2003), which chooses which heuristic to compute at each state. It may seem like the best heuristic to choose for each state is the one which yields the more accurate estimate for that state. Choosing the most accurate heuristic for each state yields a heuristic which is as accurate as the point-wise maximum of the heuristics, while still computing only one heuristic per state. As the maximum of several heuristics is at least as accurate as any of these heuristics, using their maximum leads to less expanded states (Pearl, 1984), and thus seemingly reduces overall search time.

This analysis, however, does not take into account the fact that different heuristics have different computation times. Consider the following example. We have two heuristics, h_1 and h_2, and we know that h_2 dominates h_1. A priori, it seems like using h_2 should always be preferred to using h_1 because the former will provably cause A^* to expand less states. However, suppose that on a given planning task, A^* expands 1000 states when guided by h_1 and only 100 states when guided by h_2. If computing h_1 for each state takes 10 ms, and computing h_2 for each state takes 1000 ms, then switching from h_1 to h_2 *increases* the overall search time. Using the maximum of h_1 and h_2 will only hurt, as h_2 dominates h_1, and thus computing the maximum simply wastes the time spent on computing h_1.

It is possible, however, that computing h_2 for a few carefully chosen states, and computing h_1 for all other states would result in expanding 100 states, while reducing the overall search time when compared to running A^* with only h_2. To capture this intuition, we begin by analyzing an idealized model of the search space, and describe how an all-knowing oracle would choose the best heuristic to compute for each state, in order to reduce the overall search time. Although we can not use such an all-knowing oracle in practice, we use the oracle's decision as a target concept for a classifier. At a high level, selective max can be seen as an online active learning procedure for such a classifier, as well as a procedure for choosing which heuristic to compute at each state, based on the classifier's predictions.

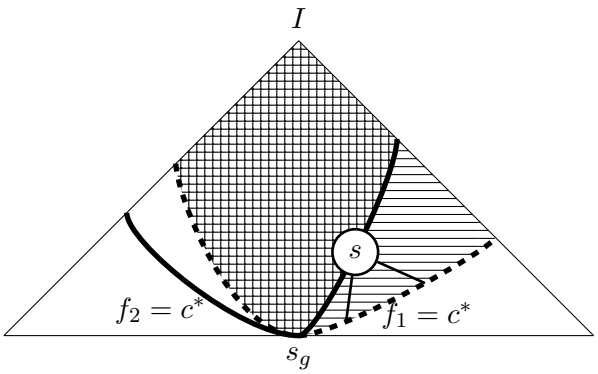


Figure 5.1: An illustration of the idealized search space model and the f-contours of two admissible heuristics.

5.2 A Model for Heuristic Selection

Given a set of admissible heuristics and the objective of minimizing the overall search time, we are interested in how an oracle, which has access to all the information it requires, would choose which heuristic to compute at each search state. It might seem that searching for an optimal solution when such an all-knowing oracle exists is unnecessary, because the oracle already knows the optimal solution. However, consider the problem of *proving* that the solution such an oracle provides is optimal. Since A^* is known to find an optimal solution, we can use the oracle to guide an A^* search, which will consume as little computation time as possible.

In what follows, we describe the oracle decisions for a pair of classical heuristics, with respect to an idealized search space model corresponding to a tree-structured search space with a single goal state, constant branching factor b, and uniform cost actions. Such an idealized search space model was used in the past to analyze the behavior of A^* (Pearl, 1984). Two additional assumptions we make are that the heuristics are consistent, and that the time t_i required for computing heuristic h_i is independent of the state being evaluated; w.l.o.g. we assume $t_2 \geq t_1$. Obviously, most of the above assumptions do not hold in typical search problems, and later we carefully examine their individual influence on our framework.

5.2.1 Idealized Model

Adopting the standard notation, let $g(s)$ be the cost of the cheapest path from I to s. Defining $\max_h(s) = \max(h_1(s), h_2(s))$, we then use the notation

$f_1(s) = g(s) + h_1(s)$, $f_2(s) = g(s) + h_2(s)$, and $\max_f(s) = g(s) + \max_h(s)$. The A^* algorithm with a consistent heuristic h expands states in increasing order of $f = g+h$. Assuming the goal state is at depth c^*, let us consider the states satisfying $f_1(s) = c^*$ (the dotted line in Fig. 5.1) and those satisfying $f_2(s) = c^*$ (the solid line in Fig. 5.1). The states above the $f_1 = c^*$ and $f_2 = c^*$ contours are those that are surely expanded by A^* with h_1 and h_2, respectively. The states above both these contours (the grid-marked region in Fig. 5.1), that is, the states $SE = \{s \mid \max_f(s) < c^*\}$, are those that are surely expanded by A^* using $\max_h$ (Pearl, 1984, Theorem 4, p. 79).

Under the objective of minimizing the search time, observe that the optimal decision for any state $s \in SE$ is not to compute any heuristic at all, since all these states are surely expanded anyway. The optimal decision for all other states is a bit more complicated. Without loss of generality, consider the states where $f_1(s) < c^*$ and $f_2(s) = c^*$; in Fig. 5.1, these are the states on the part of the $f_2 = c^*$ contour that separates between the grid-marked and lines-marked areas. Since $f_1(s)$ and $f_2(s)$ account for the same $g(s)$, we have $h_2(s) > h_1(s)$, that is, h_2 is more accurate in state s than h_1. If we were interested solely in reducing state expansions, then h_2 would obviously be the right heuristic to compute at s. However, for our objective of reducing the actual search time, h_2 may actually be the wrong choice because it might be much more expensive to compute than h_1.

Let us consider the effects of each of our two alternatives. If we compute $h_2(s)$, then s is no longer surely expanded, as $f_2(s) = c^*$, and thus whether A^* expands s or not depends on tie-breaking. As the oracle has perfect knowledge, we can assume that ties are broken favorably. In contrast, if we compute $h_1(s)$, then s is surely expanded because $f_1(s) < c^*$. Note that not computing h_2 for s and then computing h_2 for one of the descendants s' of s is surely a sub-optimal strategy as we do pay the cost of computing h_2, yet the pruning of A^* is limited only to the search sub-tree rooted in s'. Therefore, our choices are really either computing h_2 for s, or computing h_1 for all the states in the sub-tree rooted in s that lie on the $f_1 = c^*$ contour. Suppose we need to expand l complete levels of the state space from s to reach the $f_1 = c^*$ contour. This means we need to generate order of b^l states, and then invest $b^l t_1$ time in calculating h_1 for all these states that lie on the $f_1 = c^*$ contour. In contrast, suppose we choose to compute $h_2(s)$. With favorable tie-breaking, the time required to "explore" the sub-tree rooted in s will be t_2.

Putting things together, the optimal decision in state s is thus to com-

pute h_2 iff $t_2 < b^l t_1$, or if we rewrite this, if

$$l > \log_b(\frac{t_2}{t_1}).$$

As a special case, if both heuristics take the same time to compute, this decision rule boils down to $l > 0$, that is, the optimal choice is simply the more accurate (for state s) heuristic.

The next step is to somehow estimate the "depth to go" l. For that, we make another assumption about the rate at which f_1 grows in the sub-tree rooted at s. Although there are many possibilities here, we will look at two estimates that appear to be quite reasonable. The first estimate assumes that the h_1 value remains constant in the subtree rooted at s, that is, the additive error of h_1 increases by 1 for each level below s. In this case, f_1 increases by 1 for each expanded level of the sub-tree (because h_1 remains the same, and g increases by 1), and it will take expanding $\Delta_h(s) = h_2(s) - h_1(s)$ levels to reach the $f_1 = c^*$ contour. The second estimate we examine assumes that the absolute error of h_1 remains constant, that is, h_1 increases by 1 for each level expanded, and so f_1 increases by 2. In this case, we will need to expand $\Delta_h(s)/2$ levels. This can be generalized to the case where the estimate h_1 increases by any constant additive factor c, which results in $\Delta_h(s)/(c+1)$ levels being expanded. In either case, the dependence of l on $\Delta_h(s)$ is linear, and thus our decision rule can be reformulated to compute h_2 if

$$\Delta_h(s) > \alpha \log_b(\frac{t_2}{t_1}),$$

where α is a hyper-parameter for our algorithm. Note that, given b, t_1, and t_2, the quantity $\alpha \log_b(t_2/t_1)$ becomes fixed and in what follows we denote it simply by *threshold* τ.

5.2.2 Dealing with Model Assumptions

The idealized model above makes several assumptions, some of which appear to be very problematic to meet in practice. Here we examine these assumptions more closely, and when needed, suggest pragmatic compromises.

First, the model assumes that the search space forms a tree with a single goal state, and that the heuristics in question are state-dependent and consistent. Although the first assumption does not hold in most planning tasks, and the second assumption is not satisfied by some state of the art heuristics, they do not prevent us from using the decision rule suggested by the model. Furthermore, there is some empirical evidence to support our

conclusion about exponential growth of the search effort as a function of heuristic error, even when the assumptions made by the model do not hold. In particular, the experiments of Helmert and Röger (2008) on IPC benchmarks with heuristics with small constant additive errors clearly show that the number of expanded nodes typically grows exponentially as the (still very small and additive) error increases. Another assumption of our model is that all actions are unit cost. While this assumption does not prevent us from using our decision rule as is, in Section 5.3.4 we explain how we deal with planning tasks with non-uniform action costs.

The model also assumes that both the branching factor and the heuristic computation times are constant across the search states. In our application of the decision rule to planning in practice, we deal with this assumption by adopting the average branching factor and heuristic computation times, estimated from a random sample of search states. Finally, the model assumes perfect knowledge about the surely expanded search states. In practice, this information is obviously not available. We approach this issue conservatively by treating all the examined search states as if they were on the decision border, and thus apply the decision rule at all the search states. Note that this does not hurt the correctness of our algorithm, but only costs us some heuristic computation time on the surely expanded states.

5.3 Online Learning of the Selection Rule

Our decision rule for choosing a heuristic to compute at a given search state s suggests to compute the more expensive heuristic h_2 when $h_2(s) - h_1(s) > \tau$. However, computing $h_2(s) - h_1(s)$ requires computing both heuristic estimates for state s, defeating the whole purpose of reducing search time by selectively evaluating only one heuristic at each state. To overcome this pitfall, we take our decision rule as a target concept, and suggest an *active online learning* procedure for that concept. Intuitively, our concept is the set of states where the more expensive heuristic h_2 is "significantly" more accurate than the cheaper heuristic h_1. According to our model, this corresponds to the states where the reduction in expanded states by computing h_2 outweighs the extra time needed to compute it.

In what follows, we begin by giving a general overview of our learning procedure, and then describe several alternatives for each of its components. First, note that the target concept $h_2(s) - h_1(s) > \tau$ is a binary predicate. We simulate an oracle for the decision rule by training a binary classifier on the target concept. To build a classifier, we first need to collect train-

evaluate(s)
$\langle h, confidence\rangle :=$ CLASSIFY(s, *model*)
if $(confidence > \rho)$ **then**
  return $h(s)$
else
  $label := h_1$
  if $h_2(s) - h_1(s) > \alpha \log_b(t_2/t_1)$ **then** $label := h_2$
  update *model* with $\langle s, label\rangle$
  return $\max(h_1(s), h_2(s))$

Figure 5.2: The *selective max* state evaluation procedure.

ing examples, which should be representative of the entire search space — several state space sampling methods are discussed in Section 5.3.1. After the training examples T are collected, they are first used to estimate b, t_1 and t_2 by averaging the respective quantities over T. Once b, t_1 and t_2 are estimated, we can compute the threshold $\tau = \alpha \log_b(t_2/t_1)$ for our decision rule. We generate a label for each training example by calculating $\Delta_h(s) = h_2(s) - h_1(s)$, and comparing it to the decision threshold: If $\Delta_h(s) > \tau$, we label s with h_2, otherwise with h_1. If $t_1 > t_2$ we simply switch between the heuristics — our decision is always *whether to compute the more expensive heuristic or not*; the default is to compute the cheaper heuristic, unless the classifier says otherwise. Of course, to train a classifier we need to extract a set of features for each training example. The features we use are discussed in Section 5.3.2.

Once we have our training set and features to represent the examples, we can build a binary classifier for our concept. This classifier can then play the role of our hypothetical oracle, indicating which heuristic to compute where. However, as our classifier is not likely to be a perfect oracle, we further consult the confidence the classifier associates with its classification. The resulting state evaluation procedure of selective max is depicted in Figure 5.2. When state s is to be evaluated, we use our classifier to decide which heuristic to compute. If the classification confidence exceeds a confidence threshold ρ, a parameter of selective max, then only the indicated heuristic is computed for s. Otherwise, we conclude that there is not enough information to make a selective decision for s, and compute the regular maximum over $h_1(s)$ and $h_2(s)$. However, we use this opportunity to improve the quality of our prediction for states similar to s, and update our classifier, by generating a label based on $h_2(s) - h_1(s)$, and learning from the newly labeled example. To avoid the problems associated with concept drift, we

do not change the estimates for b, t_1 and t_2, so the threshold τ remains fixed. In any case, these decisions to dedicate computation time to obtain a label for a new example constitute the active part of our learning procedure. The choice of a classifier for our approach is discussed in Section 5.3.3.

5.3.1 State Space Sampling

One option for collecting the training examples is to use the first t states of the search where t is the desired number of training examples. However, this method has a bias towards states that are closer to the initial state, and therefore is not likely to represent the search space well. Hence, we instead collect training examples by sending "probes" from the initial state. Each such "probe" simulates a stochastic hill-climbing search, which terminates after reaching some depth limit. All the states generated by such a probe are used as training examples, and we stop probing after collecting t training examples.

In our evaluation, the probing depth limit was set to twice the heuristic estimate of the initial state, that is $2 \cdot \max_h(I)$, and the next state s for an ongoing probe was chosen with a probability proportional to $1/\max_h(s)$. This "inverse heuristic" selection biases the sample towards states with lower heuristic estimates, that is, to states that are more likely to be expanded during the search. It is also possible not to use the "inverse heuristic" selection bias, and simply use unbiased probes to perform a sample of the state space.

Other, more sophisticated, procedures for search space sampling have been proposed in the literature (Haslum et al., 2007). This sampling method, which we call the PDB sampling method, uses unbiased random walks (probes), but only adds the last state reached in each probe. The depth of each probe is distributed binomially around the estimated goal depth.

5.3.2 Features

Having obtained a set of training examples, we must decide about the features we use to characterize each example. Since our target concept is based on heuristic values, the features should contain the information that heuristics are derived from — typically the problem description and the current state.

While numerous features for characterizing states of planning tasks have been proposed in previous literature (Yoon et al., 2008; de la Rosa et al., 2008), they were all designed for inter-problem learning, that is, for learning

from different planning tasks. However, in our approach, we are only concerned with one problem, and our features should be suitable for learning from different states of the same problem.

In our implementation, we use the simplest features possible, taking each state variable as a feature. One important note is that although we have described the STRIPS formalism for planning tasks, our implementation uses the SAS$^+$ formalism internally. The SAS$^+$ formalism uses finite domain variables — each variable is not just a $TRUE/FALSE$ binary proposition, but has a finite domain of possible values.

A SAS$^+$ description of a planning task can be automatically generated from a STRIPS-like description (Helmert, 2009b). This is done by synthesizing invariants of the planning domain, in order to identify groups of mutually exclusive binary propositions. A group of mutually exclusive propositions can be represented as a single SAS$^+$ variable. Thus, we have a small number of finite domain variables, rather than a large number of binary variables.

5.3.3 Classifier

The last decision to be made is the choice of classifier. Although many classifiers can be used here, there are several requirements that need to be met due to our particular setup. First, both training and classification must be very fast, as both are performed during time-constrained problem solving. Second, the classifier must be incremental to allow online update of the learned model. Finally, the classifier should provide us with a meaningful confidence for its predictions.

While several classifiers meet these requirements, we found the classical Naive Bayes classifier to provide a good balance between speed and accuracy. One note on the Naive Bayes classifier is that it assumes a very strong conditional independence between the features. Although this is not a fully realistic assumption for planning tasks, using a SAS$^+$ task formulation in contrast to the classical STRIPS formulations helps a lot: instead of many binary variables which are highly dependent upon each other, we have a much smaller set of variables which are less dependent upon each other.

While Naive Bayes is the most suitable classifier for use with selective max, it is also possible to use other classifiers. The most obvious choice for a replacement classifier is another Bayesian classifier. One such classifier is AODE (Webb et al., 2005), an extension of the Naive Bayes classifier, which somewhat relaxes the assumption of independence between the features, and is typically more accurate than Naive Bayes. However, this added accuracy comes at the cost of increased training and classification time.

Decision trees are another popular type of classifier, which allows for even faster classification. Although most decision tree induction algorithms are not incremental, and thus are not suitable for our setting, the Incremental Tree Inducer (ITI) algorithm (Utgoff et al., 1997) supports incremental updating of decision trees by tree restructuring, and furthermore, has a freely available implementation in C. In our evaluation, we used ITI in incremental mode, and incorporated every example into the tree immediately, because the tree will usually be used for many classifications between adding two consecutive training examples. The classification confidence with the ITI classifier is obtained by the frequency of examples at the leaf node from which the classification came from.

A different family of classifiers which could be used is k-Nearest Neighbors (kNN) (Cover and Hart, 1967). In order to use kNN, we need a distance metric between examples, which, with the above features, are simply states. As with our choice of features, we opt for simplicity, and use L_2 as our distance metric. kNN enjoys very fast learning time, but suffers from expensive classification time. The classification confidence is obtained by a simple (unweighted) vote between the k nearest neighbors.

Another question, related to the choice of classifier is feature selection. In some planning tasks, the number of variables (that is, features) can be over 2000 (for example, task 35 of the AIRPORT domain has 2558 variables). While it is likely that using feature selection will improve the performance of Naive Bayes and kNN, doing so poses a problem when the initial sample is considered. Since feature selection will have to be done right after the initial sample is obtained, it will have to be based only on the initial sample. This might cause a problem since some features might appear to be irrelevant according to the initial sample, yet turn out to be very relevant when active learning is used after some low-confidence states are encountered. Therefore, we do not use feature selection in our implementation of selective max.

5.3.4 Handling Non-Uniform Action Costs

One of the assumptions of our theoretical model is that all actions are unit cost. However, some planning tasks feature varying action costs. Notably, all planning tasks from the 2008 International Planning Competition feature non-unit action costs. Since our model assumes that the heuristic value is the same as the depth to go, this could lead to some erroneous decisions. Therefore, when dealing with non unit-cost tasks, we also estimate the cost of the average action from our initial state space sample, and convert heuristic estimates of cost-to-go into heuristic estimates of depth-to-go by dividing

the cost-to-go estimate by the average action cost. We do this by adapting our threshold, so that if the average action cost is $\hat{c}$, our concept becomes $\frac{h_2(s)-h_1(s)}{\hat{c}} > \tau$.

Another place where varying action costs pose a problem is in the state space sample. When faced with non unit-cost tasks, we can no longer use $2 \cdot \max_h(I)$ as our goal *depth* estimate, because the heuristics estimate cost, not depth. We, however, want to obtain an estimate of the depth of the cheapest plan. Thus, computing a heuristic estimate for a modified task, with all actions being unit-cost, is not the right thing to do, as this would yield an estimate of the shallowest plan.

Therefore, we estimate the depth of the cheapest goal, by using a heuristic on the original task, and then use the number of actions which the heuristic accounts for as our goal depth estimate. While this is not possible with any given heuristic, in our implementation we use the monotonically-relaxed plan heuristic, also known as FF heuristic (Hoffmann and Nebel, 2001), in such a way. We first use the heuristic to find a relaxed plan from the initial state, and then simply take the number of actions in the relaxed plan as our goal depth estimate.

5.3.5 Extension to Multiple Heuristics

As a final note, extending selective max to use more than two heuristics is rather straightforward — simply compare the heuristics in a pair-wise manner, and choose the best heuristic by a vote, which can either be a regular vote (i.e., 1 for the winner, 0 for the loser), or weighted according to the classifier's confidence. Although this requires a quadratic number of classifiers, training and classification time (at least with Naive Bayes) appear to be much lower than the overall time spent on heuristic computations, and thus the overhead induced by learning and classification is likely to remain relatively low for non-negligible ensembles of heuristics.

5.4 Related Work

Learning for planning has been a very active field starting the early days of planning (Fikes et al., 1972), and is recently receiving growing attention in the community. So far, however, relatively little work has dealt with learning for heuristic search planning, one of the most prominent approaches to planning these days. Most works in this direction have been devoted to learning macro-actions (Finkelstein and Markovitch, 1998; Botea et al., 2005; Coles and Smith, 2007). Among the other works, the one most closely

related to ours is probably the work by Yoon et al. (2008) that suggest learning an (inadmissible) heuristic function based upon features extracted from relaxed plans. In contrast, our focus is on optimal planning. Overall, we are not aware of any previous work that deals with learning for optimal heuristic search.

Chapter 6

Empirical Evaluation

So far, we have discussed ideas which are of theoretical interest. In this chapter, we present an empirical evaluation, which demonstrates that these ideas also lead to state of the art performance in cost-optimal planning.

We conducted an empirical evaluation on problems from all 31 STRIPS domains from IPC 1998–2008. The evaluation was conducted on a single core of an Intel E8400 3.0 Ghz CPU, with a memory limit of 6 GB, and a time limit of 30 minutes, on a 64-bit Linux system.

6.1 Search Algorithm Evaluation

We have discussed three different search algorithms in this work: the well known A^* search algorithm, MPD-A*, which we claim is better adapted to using multi-path dependent heuristics, and *path-*A^*, which is suitable for path admissible heuristics. In order to demonstrate our claim that MPD-A* is better adapted to using multi-path dependent heuristics, we used the same multi-path dependent heuristic — h_{LA} with optimal cost partitioning, varying the search algorithm. The landmarks the heuristic used were all single fact landmarks of the delete relaxation, as discovered by the procedure of Keyder et al. (2010).

Table 6.1 presents the number of problems solved in each domain, and in total, using the different search algorithms. As the results show, MPD-A* solved more problems overall than both A^* and *path-*A^*, by quite a wide margin. Tables 6.2 and 6.3 show the total number of expanded states, and the geometric mean of total solution time, respectively. Both are only over the problems solved by all three configurations. As these table show, MPD-A* expands less states in total, and is faster on average, than both other

coverage	**MPD-A***	*A**	*path-A**
airport (50)	28	28	28
blocks (35)	**21**	17	17
depot (22)	4	4	4
driverlog (20)	7	7	7
elevators-opt08-strips (30)	7	7	7
freecell (80)	51	**52**	51
grid (5)	2	2	2
gripper (20)	5	5	5
logistics00 (28)	**20**	10	10
logistics98 (35)	**3**	2	2
miconic (150)	141	141	141
mprime (35)	15	15	15
mystery (30)	12	12	12
openstacks-opt08-strips (30)	12	12	12
parcprinter-08-strips (30)	11	11	11
pathways (30)	4	4	4
pegsol-08-strips (30)	26	26	26
pipesworld-notankage (50)	**15**	14	14
pipesworld-tankage (50)	9	**10**	**10**
psr-small (50)	48	48	48
rovers (40)	5	5	5
satellite (36)	4	4	4
scanalyzer-08-strips (30)	13	13	13
sokoban-opt08-strips (30)	15	**16**	**16**
storage (30)	13	13	13
tpp (30)	5	5	5
transport-opt08-strips (30)	9	9	9
trucks-strips (30)	6	6	6
woodworking-opt08-strips (30)	11	11	11
zenotravel (20)	8	8	8
SUM	**530**	517	516

Table 6.1: Number of problems solved from each domain, using different search algorithms with h_{LA}

expansions	**MPD-A***	*A**	*path-A**
airport (28)	**307635**	313257	313257
blocks (17)	**21026**	256810	256810
depot (4)	**401684**	2028129	2028129
driverlog (7)	363541	363541	363541
elevators-opt08-strips (7)	**483728**	502436	502436
freecell (51)	**511735**	533939	533939
grid (2)	**467076**	467534	467534
gripper (5)	458498	458498	458498
logistics00 (10)	**3472**	1012746	1012746
logistics98 (2)	**17021**	315239	315239
miconic (141)	135213	135213	135213
mprime (15)	**313576**	313579	313579
mystery (14)	290133	290133	290133
openstacks-opt08-strips (12)	1579931	1579931	1579931
parcprinter-08-strips (11)	158090	158090	158090
pathways (4)	173593	173593	173593
pegsol-08-strips (26)	**3949270**	4163429	4163429
pipesworld-notankage (14)	**1346551**	1441610	1441610
pipesworld-tankage (9)	**319465**	430821	430821
psr-small (48)	698003	698003	698003
rovers (5)	**231380**	248852	248852
satellite (4)	**10623**	10987	10987
scanalyzer-08-strips (13)	23213	23213	23213
sokoban-opt08-strips (15)	**3462312**	3486061	3486061
storage (13)	**474921**	482364	482364
tpp (5)	12355	12355	12355
transport-opt08-strips (9)	929285	929285	929285
trucks-strips (6)	**1261600**	1351763	1351763
woodworking-opt08-strips (11)	**151777**	236626	236626
zenotravel (8)	186334	186334	186334
SUM	**18743041**	22604371	22604371

Table 6.2: Total number of expansions in each domain, for the problems solved by all configurations (in parentheses)

total_time	**MPD-A***	A^*	*path-*A^*
airport (28)	0.8249	**0.8136**	0.8179
blocks (17)	**0.2899**	0.354	0.3651
depot (4)	**7.102**	9.2569	9.8935
driverlog (7)	5.2367	**4.1963**	4.3627
elevators-opt08-strips (7)	52.6276	**45.7445**	51.1003
freecell (51)	6.7982	**6.4825**	6.7045
grid (2)	18.8532	**17.3217**	17.3554
gripper (5)	2.5272	**1.585**	2.0326
logistics00 (10)	**0.1651**	1.0814	1.3887
logistics98 (2)	**3.1769**	17.1702	24.8938
miconic (141)	0.5529	0.5496	**0.5494**
mprime (15)	10.0974	**8.9162**	8.9257
mystery (14)	1.9163	**1.7599**	1.7712
openstacks-opt08-strips (12)	12.6003	9.0738	**9.0629**
parcprinter-08-strips (11)	0.5221	**0.4546**	0.4557
pathways (4)	2.1029	1.4368	**1.4148**
pegsol-08-strips (26)	5.0574	**4.2123**	4.4027
pipesworld-notankage (14)	**1.8978**	2.7356	2.7938
pipesworld-tankage (9)	**3.1116**	5.1286	5.2625
psr-small (48)	0.3559	0.329	**0.3289**
rovers (5)	0.4287	**0.3935**	0.3937
satellite (4)	0.5022	**0.4395**	0.5063
scanalyzer-08-strips (13)	1.8513	**1.7838**	1.9973
sokoban-opt08-strips (15)	5.929	**5.2458**	5.2709
storage (13)	0.8202	**0.7919**	0.7951
tpp (5)	0.1945	**0.1838**	**0.1838**
transport-opt08-strips (9)	3.3152	**2.6075**	2.7001
trucks-strips (6)	39.9974	**31.3528**	31.9196
woodworking-opt08-strips (11)	**2.5245**	2.5318	2.7453
zenotravel (8)	1.924	**1.602**	1.6296
GEOMETRIC MEAN	**2.2357**	2.3475	2.4808

Table 6.3: Geometric mean of total solution time in seconds, over the problems solved by all configurations (in parentheses)

search algorithms.

6.2 Evaluation of Existential Optimal Landmarks

We have seen that MPD-A* is indeed better when using the multi-path dependent h_{LA} heuristic than A^* and *path-A**. However, we have also described $\exists$-opt landmarks, which make the resulting heuristic path-admissible, and thus unsuitable for use with MPD-A*. Therefore, we have performed an empirical evaluation to check whether the added information derived from $\exists$-opt landmarks is worth the need to switch to *path-A**.

We tested the two variants of $\exists$-opt landmark formulae discussed previously: $\phi_{\mathcal{L}}(\pi)$ and $\phi_{\mathcal{L}}(\pi|\{\rho\})$. Recall that while $\phi_{\mathcal{L}}(\pi)$ is path admissible, $\phi_{\mathcal{L}}(\pi|\{\rho\})$ is $\{\rho\}$-path admissible. Consequently, with $\phi_{\mathcal{L}}(\pi|\{\rho\})$ we use the modified version of *path-A**, discussed in Section 4.4. Thus, with $\phi_{\mathcal{L}}(\pi)$ we only compute a new heuristic estimate for state s when a cheaper path to s has been found. With $\phi_{\mathcal{L}}(\pi|\{\rho\})$ we compute a new heuristic estimate also when a new path π' to s of the same cost as the current path π has been found, and this if π' is lexicographically lower than π. We also compare to *path-A** using the regular h_{LA} heuristic.

Table 6.4 presents the number of problems solved in each domain, and in total, using the different configurations. As the results show, using the path admissible $\exists$-opt variant $\phi_{\mathcal{L}}(\pi)$ solved more problems overall than both the $\{\rho\}$-path admissible $\exists$-opt variant $\phi_{\mathcal{L}}(\pi|\{\rho\})$ and than regular h_{LA}. We believe this is because $\phi_{\mathcal{L}}(\pi|\{\rho\})$ prunes too many optimal solutions, while h_{LA} does not provide enough pruning power. We also list, for reference, the number of problems solved by MPD-A* with the h_{LA} heuristic. As the table shows, *path-A** with $\phi_{\mathcal{L}}(\pi)$ solves more problems than MPD-A* with h_{LA}, allowing us to conclude that $\exists$-opt landmarks are indeed useful in practice.

Tables 6.2 and 6.3 show the total number of expanded states, and the geometric mean of total solution time, respectively. Both are only over the problems solved by all three configurations. As these table show, *path-A** with $\phi_{\mathcal{L}}(\pi)$ expands less states in total, and is faster on average, than *path-A** with the other two heuristics.

coverage	$\phi_{\mathcal{L}}(\pi)$	$\phi_{\mathcal{L}}(\pi\|\{\rho\})$	h_{LA}	**MPD-A***
airport (50)	**28**	27	**28**	28
blocks (35)	17	17	17	21
depot (22)	4	4	4	4
driverlog (20)	**9**	**9**	7	7
elevators (30)	6	0	**7**	7
freecell (80)	**51**	49	**51**	51
grid (5)	2	2	2	2
gripper (20)	5	5	5	5
logistics00 (28)	**20**	**20**	10	20
logistics98 (35)	**3**	**3**	2	3
miconic (150)	141	141	141	141
mprime (35)	**19**	17	15	15
mystery (30)	**15**	**15**	12	12
openstacks (30)	12	12	12	12
parcprinter (30)	**12**	**12**	11	11
pathways (30)	4	4	4	4
pegsol (30)	26	26	26	26
pipesworld-notankage (50)	14	14	14	15
pipesworld-tankage (50)	**10**	7	**10**	9
psr-small (50)	48	48	48	48
rovers (40)	5	5	5	5
satellite (36)	**6**	**6**	4	4
scanalyzer (30)	13	13	13	13
sokoban (30)	**16**	0	**16**	15
storage (30)	**14**	**14**	13	13
tpp (30)	**6**	**6**	5	5
transport (30)	9	9	9	9
trucks-strips (30)	**7**	**7**	6	6
woodworking (30)	11	11	11	11
zenotravel (20)	8	8	8	8
SUM	**541**	511	516	530

Table 6.4: Number of problems solved from each domain, using different landmarks

expansions	$\phi_{\mathcal{L}}(\pi)$	$\phi_{\mathcal{L}}(\pi \mid \{\rho\})$	h_{LA}
airport (27)	**212128**	423126	212786
blocks (17)	**243021**	259587	256810
depot (4)	**406646**	554578	2028129
driverlog (7)	**160793**	207027	363541
freecell (49)	**425035**	582345	425234
grid (2)	**232425**	240574	467534
gripper (5)	**458498**	594875	**458498**
logistics00 (10)	**1655**	2221	1012746
logistics98 (2)	**5954**	8714	315239
miconic (141)	**135213**	183319	**135213**
mprime (15)	**32427**	33004	313579
mystery (14)	**37967**	44219	290133
openstacks (12)	**1579931**	1756117	**1579931**
parcprinter (11)	**101178**	146959	158090
pathways (4)	**32287**	39634	173593
pegsol (26)	**3670923**	4001486	4163429
pipesworld-notankage (14)	**1227678**	1747983	1441610
pipesworld-tankage (7)	**18658**	34377	25881
psr-small (48)	**361394**	373886	698003
rovers (5)	**104511**	353604	248852
satellite (4)	**6112**	9488	10987
scanalyzer (13)	**22157**	26880	23213
storage (13)	**320586**	357176	482364
tpp (5)	**4227**	7355	12355
transport (9)	**912637**	1018792	929285
trucks-strips (6)	**244408**	323278	1351763
woodworking (11)	**89434**	151931	236626
zenotravel (8)	**62681**	73426	186334
SUM	**11110564**	13555961	18001758

Table 6.5: Total number of expansions in each domain, for the problems solved by all configurations (in parentheses)

total_time	$\phi_{\mathcal{L}}(\pi)$	$\phi_{\mathcal{L}}(\pi \mid \{\rho\})$	h_{LA}
airport (27)	1.1419	1.3844	**0.6647**
blocks (17)	0.4095	0.4194	**0.3651**
depot (4)	**7.188**	9.9127	9.8935
driverlog (7)	**4.0427**	5.8268	4.3627
freecell (49)	7.4946	8.6938	**5.497**
grid (2)	**16.4767**	17.053	17.3554
gripper (5)	2.6924	3.9099	**2.0326**
logistics00 (10)	**0.1332**	0.1436	1.3887
logistics98 (2)	**1.7126**	3.4551	24.8938
miconic (141)	0.7473	0.7494	**0.5494**
mprime (15)	**3.0758**	4.5382	8.9257
mystery (14)	**0.8707**	0.9833	1.7712
openstacks-opt08-strips (12)	10.5737	15.1019	**9.0629**
parcprinter-08-strips (11)	0.4791	0.5605	**0.4557**
pathways (4)	**0.8625**	1.2494	1.4148
pegsol-08-strips (26)	4.5828	5.2247	**4.4027**
pipesworld-notankage (14)	**2.775**	3.5011	2.7938
pipesworld-tankage (7)	1.3249	1.8142	**1.1401**
psr-small (48)	**0.3109**	0.3371	0.3289
rovers (5)	**0.3927**	0.4832	0.3937
satellite (4)	0.5391	0.6481	**0.5063**
scanalyzer-08-strips (13)	2.1372	2.2772	**1.9973**
storage (13)	0.9239	1.0368	**0.7951**
tpp (5)	**0.1754**	0.1985	0.1838
transport-opt08-strips (9)	4.1689	5.1784	**2.7001**
trucks-strips (6)	**11.0414**	15.1125	31.9196
woodworking-opt08-strips (11)	**2.7081**	3.0248	2.7453
zenotravel (8)	**1.5274**	1.9756	1.6296
GEOMETRIC MEAN	**1.6137**	1.9982	2.0228

Table 6.6: Geometric mean of total solution time in seconds, over the problems solved by all configurations (in parentheses)

6.3 Selective Max Evaluation

coverage	h_{LA}	$h_{\text{LM-CUT}}$	sel_h
barman	4	4	4
elevators	14	**18**	**18**
floortile	2	**7**	**7**
nomystery	**20**	15	**20**
openstacks	14	**16**	14
parcprinter	11	**13**	**13**
parking	3	2	**4**
pegsol	17	**18**	17
scanalyzer	6	**12**	10
sokoban	20	20	20
tidybot	14	14	14
transport	**7**	6	6
visitall	10	10	10
woodworking	9	**12**	**12**
SUM	151	167	**169**

Table 6.7: Number of planning tasks solved at IPC 2011.

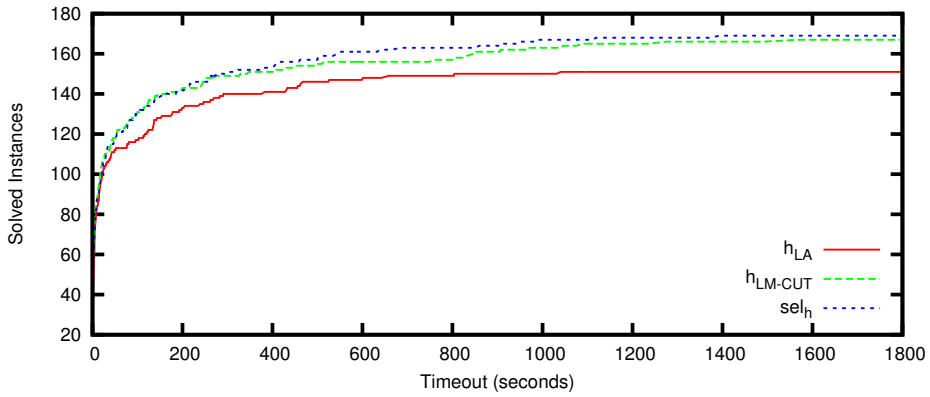


Figure 6.1: IPC-2011: Anytime performance in terms of coverage.

The last results we report are, in fact, results from the last International Planning Competition — IPC-2011. The IPC is run by a third party (García-Olaya et al., 2011), and is, arguably, the most impartial evaluation method for planning systems. This edition of the IPC included 14 domains, several of which were completely new — a fact which precludes any offline learning approach.

A version of Fast Downward using MPD-A* with selective max over h_{LA}

(using uniform cost partitioning) and $h_{\text{LM-CUT}}$ (Helmert and Domshlak, 2009), as well as independent versions of Fast Downward based on these two heuristics individually, participated at the sequential optimal track of IPC-2011. Selective max was the runner-up ex-aequo at IPC-2011, tying for 2nd place with a version of Fast Downward using an abstraction "merge-and-shrink" heuristic (Nissim et al., 2011), and losing to a sequential portfolio combining the heuristics used in both runners-up (Helmert et al., 2011).

Table 6.7 shows the number of tasks solved in each domain of IPC-2011, and Figure 6.1 shows the anytime profile of these 3 planners on IPC-2011 tasks, plotting the number of tasks solved under different timeouts. These times do include the common preprocessing time, as the IPC measures overall planner performance. As the results indicate, selective max dominates each of its component heuristics under any given timeout, not just under the 30 minute timeout of the competition. In Appendix B we describe our extended empirical evaluation of selective max, using three state of the art heuristics.

Chapter 7

Conclusion

Heuristic search is one of the oldest techniques in AI. Most work on heuristic search has only considered classical, state-dependent, heuristics. Thus, when looking for an optimal solution, the combination of A^* with an admissible heuristic comes to mind almost automatically. However, this is not the only method for finding an optimal solution, even in the context of forward state-space search.

We have formally defined the notion of global admissibility, which is weaker than admissibility, yet is enough to guarantee that A^* finds an optimal solution. We also introduced the even weaker notions of path admissibility and global path admissibility, and presented appropriate search algorithms which find an optimal solution given such heuristics. While our definition of global admissibility is novel, the idea behind it has been proposed before (Dechter and Pearl, 1985), and in fact, some optimality-preserving techniques for search space pruning, such as symmetry breaking and state-space reductions (Fox and Long, 2002; Rintanen, 2003; Coles and Smith, 2008; Chen and Yao, 2009; Pochter et al., 2011), can be seen as using a globally admissible heuristic, assigning a heuristic value of ∞ to some states, despite the fact that the goal is achievable from these states.

In that respect, our work on $\exists$-opt landmarks can be seen as extending the palette of techniques, as well as (and even more importantly) the sources of information that can be used for relaxing admissibility: while the aforementioned pruning techniques are typically based on syntactic properties of the problem description (such as functional equivalence of two objects), our $\exists$-opt landmarks inference technique performs a continuous semantic analysis of information revealed by the search process. Furthermore, our technique is not limited to "black or white", prune or don't prune reason-

ing, but can distinguish between "shades of gray", assigning different (not necessarily admissilbe) heuristic estimates to different states. This type of reasoning is not possible in classical, state-dependent heuristics, as the state alone does not provide enough information.

Another way of exploiting information gather during search is selective max, a history dependent heuristic, which is a more effective method for combining arbitrary admissible heuristics than their regular point-wise maximization. Another advantage of the selective max approach is that it can successfully exploit pairs of heuristics where one dominates the other — for example, the h_{LA} heuristic with uniform and optimal action cost partitioning schemes. The heuristic induced by the optimal action cost partitioning dominates the one induced by the uniform action cost partitioning, but takes much longer to compute. Selective max could be used to learn when it is worth spending the extra time to compute the optimal cost partitioning, and when it is not. In contrast, the max-based combination of these two heuristics would simply waste the time spent on computing the uniform action cost partitioning.

The development of non-classical heuristics was also the driving force behind the development of new search algorithms, which better exploit these new heuristics. In fact, the multi-path dependent nature of landmarks was the trigger for the development of the MPD-A* search algorithm (Karpas and Domshlak, 2009), and the $\exists$-opt landmarks were the trigger for developing *path-A**.

To conclude, Table 7.1 lists all of the heuristics presented in this work, as well as what information they depend on, and their admissibility properties.

Heuristic	Information Dependence	Admissibility
h_{LA}	Multi-path	Admissible
$\phi_{\mathcal{L}}(\pi)$	Path	Path admissible
$\phi_{\mathcal{L}}(\pi\|\{\rho\})$	Path	Globally path admissible
selective max	History	Admissible

Table 7.1: Summary of heuristics and their properties

The empirical evaluation demonstrates that these heuristics are not just of theoretical interest, but state of the art tools for cost-optimal classical planning. In fact, a planner using selective max to combine h_{LA} and $h_{\text{LM-CUT}}$ (Helmert and Domshlak, 2009), another landmark based heuristic, was runner-up ex-aequo in the sequential optimal track of the 2011 edition of the International Planning Competition (García-Olaya et al., 2011).

Appendix A

h_{LA} Inconsistency Example

Here we demonstrate an example task where the same state can be reached via different paths, leading to inconsistency of h_{LA} under optimal cost partitioning, and demonstrating that A^* and MPD-A* will need to reopen some nodes.

Before we describe the example task in full, we highlight the most important part of this example. Our example task has 3 landmarks: p_1, p_2, p_3. It is possible to achieve p_3 by either going through p_1 and p_2, or by going through q (which is not a landmark). However, p_1 and p_2 must be achieved *after* reaching p_3 for the first time. Thus, by achieving p_3 by going through p_1 and p_2, these landmarks are marked as achieved, but by reaching the same state through q, p_1 and p_2 are not achieved, which causes the inconsistency. An illustration of this part of the search space can be seen in Figure A.1.

A complete description of this task is given by

- Propositions: $P = \{i_1, i_2, p_1, p_2, p_3, q, g', g\}$,
- Initial state: $I = \{i_1, i_2\}$
- Goal: $G = \{g\}$, and
- Actions (given as $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$):
  - $a_1 = \langle \{i_1, i_2\}, \{p_1\}, \{i_1\} \rangle$, $\mathcal{C}(a_1) = 1$
  - $a_2 = \langle \{p_1, i_2\}, \{p_2\}, \{p_1\} \rangle$, $\mathcal{C}(a_2) = 1$
  - $a_3 = \langle \{p_2, i_2\}, \{p_3\}, \{p_2\} \rangle$, $\mathcal{C}(a_3) = 1$
  - $a_q = \langle \{i_1, i_2\}, \{q\}, \{i_1\} \rangle$, $\mathcal{C}(a_q) = 1$

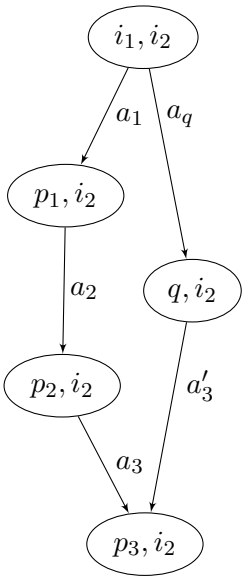


Figure A.1: Important part of search space

- $a'_q = \langle\{p_3, i_2\}, \{q\}, \{\}\rangle$, $\mathcal{C}(a'_q) = 1$
- $a'_3 = \langle\{q, i_2\}, \{p_3\}, \{q\}\rangle$, $\mathcal{C}(a'_3) = 1$
- $a'_2 = \langle\{p_3, q, i_2\}, \{p_2\}, \{\}\rangle$, $\mathcal{C}(a'_2) = 1$
- $a'_1 = \langle\{p_2, p_3, q, i_2\}, \{p_1\}, \{\}\rangle$, $\mathcal{C}(a'_1) = 1$
- $a_{g_1} = \langle\{p_1, p_2, p_3, i_2\}, \{g'\}, \{p_1, p_2, p_3, q\}\rangle$, $\mathcal{C}(a_{g_1}) = 0$
- $a_{g_2} = \langle\{i_1, i_2\}, \{p_1, p_2, p_3, g'\}, \{i_1, i_2\}\rangle$, $\mathcal{C}(a_{g_2}) = 1000$
- $a_g = \langle\{g'\}, \{g\}, \{p_1, p_2, p_3, q, g', i_1, i_2\}\rangle$, $\mathcal{C}(a_g) = 0$

The full search space is illustrated in Figure A.2, and a complete run-through of this example follows next. The trivial (that is, initial state and goal) landmarks are i_1, i_2, g. Since the only achiever of g is a_g, and $\mathsf{pre}(a_g) = \{g'\}$, g' is a landmark, and $g' \ll_{\text{gn}} g$. Since the path $\langle a_{g_2}, a_g\rangle$ is a solution which does not achieve q, q is not a landmark. All other possible solutions must achieve g' by applying a_{g_1}, and $\{p_1, p_2, p_3\} \subseteq \mathsf{pre}(a_{g_1})$. Note that the solution $\langle a_{g_2}, a_g\rangle$ achieves p_1, p_2, p_3 at the same time as g', so p_1, p_2, p_3 are landmarks, and there is no ordering between them and g'. The other orderings are the obvious natural orderings: the initial state landmarks i_1 and i_2 are ordered before p_1, p_2, p_3, g', and p_1, p_2, p_3 are ordered before the goal landmark g. The landmarks and orderings are illustrated in Figure A.3.

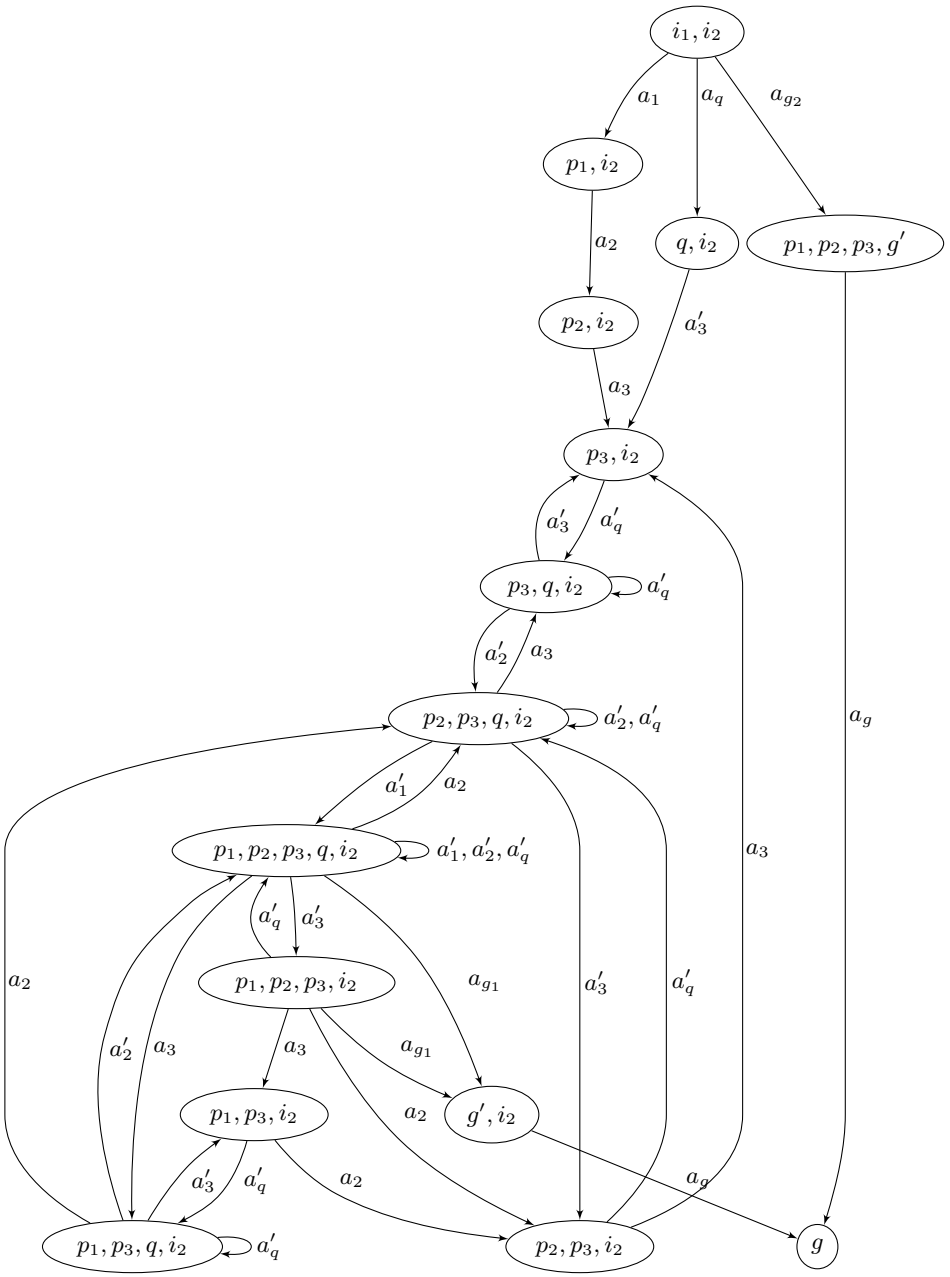


Figure A.2: Search space of example task

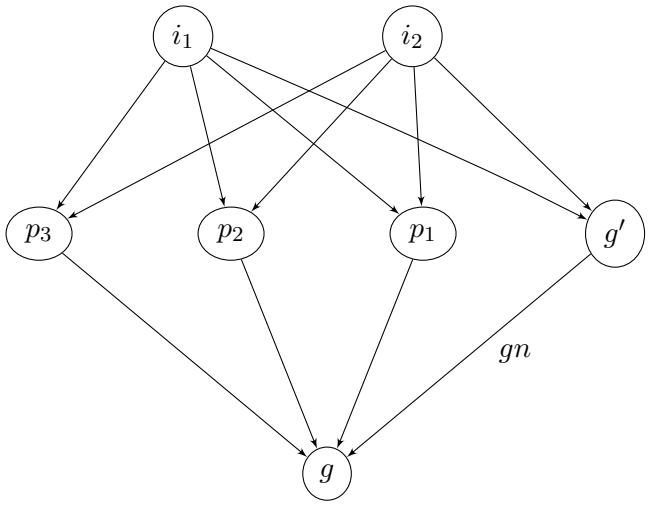


Figure A.3: Landmarks and orderings of example task

Let us now examine how A^* behaves on this example task. At the initial state $\{i_1, i_2\}$, the landmarks that need to be achieved are p_1, p_2, p_3, g', g. Both g and g' have a 0-cost achiever (a_{g_1} and a_g, respectively), and thus $cost(g) = cost(g') = 0$. The other landmarks p_1, p_2, p_3 all have separate achievers with unit costs, as well as a_{g_2} which achieves them together at a cost of 1000. Under optimal cost partitioning, $cost(p_1) = cost(p_2) = cost(p_3) = 1$, for a total heuristic estimate of 3. The successors of $\{i_1, i_2\}$ are $\{p_1, i_2\}$, $\{q, i_2\}$ and $\{p_1, p_2, p_3, g'\}$, with $g(\{p_1, i_2\}) = g(\{q, i_2\}) = 1$ and $g(\{p_1, p_2, p_3, g'\}) = 1000$. Moving to $\{p_1, i_2\}$ achieves landmark p_1, and it is easy to verify that $h_{LA}(\langle a_1 \rangle) = 2$, so $\overline{f}(\{p_1, i_2\}) = 3$. Moving to $\{q, i_2\}$ does not achieve any landmarks, so the heuristic estimate stays the same, and $\overline{f}(\{q, i_2\}) = 4$. It is also easy to see that $\overline{f}(\{p_1, p_2, p_3, g'\}) = 1000$. So the next state that will be expanded is $\{p_1, i_2\}$, which only has one successor, $\{p_2, i_2\}$. This action achieves p_2, and p_1 is still accepted, so $h_{LA}(\langle a_1, a_2 \rangle) = 1$, and $\overline{f}(\{p_2, i_2\}) = 3$. Note that although it is possible to prove that p_1 is a landmark when starting from $\{p_2, i_2\}$, this is not captured by the required again rules, and so the heuristic can not use this information. At this point, $\{p_2, i_2\}$ is expanded, and generates the only successor $\{p_3, i_2\}$. Since following this path p_1, p_2 and p_3 have been achieved, $h_{LA}(\langle a_1, a_2, a_3 \rangle) = 0$, and $\overline{f}(\{p_3, i_2\}) = 3$. Search then continues, and expands $\{p_3, q, i_2\}$ with $\overline{f}(\{p_3, q, i_2\}) = 4$, and then generates $\{p_2, p_3, q, i_2\}$ with $\overline{f}(\{p_2, p_3, q, i_2\}) = 5$. Finally, $\{q, i_2\}$, which has been on the open list

all this time is expanded, generating $\{p_3, i_2\}$ again, but this time via the path $\langle a_q, a'_3 \rangle$. Since this path is cheaper than the previous path to $\{p_3, i_2\}$, $\{p_3, i_2\}$ must be reopened. Note that the only landmark path $\langle a_q, a'_3 \rangle$ achieves is p_3, and so $h_{LA}(\langle a_q, a'_3 \rangle) = 2$, which is different from $h_{LA}(\langle a_1, a_2, a_3 \rangle) = 0$. Also note that multi-path dependence could not have been used up to this point, so using MPD-A* instead of A^* would not make any difference.

Appendix B

Selective Max Empirical Evaluation

In Section 6.3 we presented the results pertinent to selective max from IPC-2011, where selective max was a runner-up ex-aequo in the sequential optimal track. While the IPC-2011 results are an impartial evaluation of the planner that was submitted, we have also conducted our own extended empirical evaluation of selective max. In our evaluation of selective max we used three state of the art admissible heuristics: h_{LA} with uniform cost partitioning (Karpas and Domshlak, 2009), $h_{\text{LM-CUT}}$ (Helmert and Domshlak, 2009), and $h^{+}_{\text{LM-CUT}}$ (Bonet and Helmert, 2010). None of these base heuristics yields better search performance than the others across all planning domains. Of these heuristics, h_{LA} is typically the fastest to compute and the least informative, $h_{\text{LM-CUT}}$ is more expensive to compute and more informative, and $h^{+}_{\text{LM-CUT}}$ is the most expensive to compute and the most informative.[1] While there are other admissible heuristics for SAS^{+} planning that are competitive with the three heuristics above (for example, Helmert et al. (2007); Nissim et al. (2011); Katz and Domshlak (2010b)), these heuristics are based on expensive offline preprocessing, and then very fast online per-state computation. On the other hand, h_{LA}, $h_{\text{LM-CUT}}$ and $h^{+}_{\text{LM-CUT}}$ perform most of their computation online, and thus can be better exploited by selective max.

Our empirical evaluation is divided into two parts. First, we explore the performance of selective max on all benchmarks from the International Planning Competitions (IPC) 1998–2008 which the heuristics we use supported,

[1] Of course, from the least expensive to the most expensive, all three heuristics are computable in polynomial time.

Parameter	Default value	Meaning
α	1	heuristic difference bias
ρ	0.6	confidence threshold
t	100	initial sample size
Sampling Method	PDB (Haslum et al., 2007)	state space sampling method
Classifier	Naive Bayes	classifier type

Table B.1: Parameters for sel_h.

and examine the anytime performance profile of selective max. Second, we focus on a few interesting domains, and explore the effect of various parameter settings on the performance of selective max.

In the first two parts of our evaluation, the search for each planning task instance was limited to 30 minutes[2] and to 3 GB of memory. The search times do not include some preprocessing which is common to all planners, and is tangential to the issues considered in our study. The search times do include learning and classification time for selective max.

B.1 IPC 1998–2008

We begin by comparing selective max to other methods of heuristic combination on every possible choice of two or more heuristics out of h_{LA}, $h_{\text{LM-CUT}}$ and $h^+_{\text{LM-CUT}}$. We compare selective max (sel_h) to the regular maximum ($\max_h$), as well as to a planner which chooses which heuristic to compute at each state randomly (rnd_h). We performed this comparison on all 31 domains without conditional effects or axioms (which none of the heuristics we used support) from the International Planning Competitions 1998–2008.

The configuration of selective max used in all of these experiments is the default configuration described in Table B.1. In Section B.2 we explore the impact of each of these parameters on the runtime performance.

Empirical Performance

We first compare the performance of each combination method ($\max_h$, rnd_h, sel_h) on every combination of two or more heuristics out of h_{LA}, $h_{\text{LM-CUT}}$, $h^+_{\text{LM-CUT}}$. However, before we present the data for combinations of heuristics, we first present the data for individual heuristics in Table B.2. The results show that indeed, the most informed heuristic ($h^+_{\text{LM-CUT}}$) does not

[2]Each search was given a single core of a 3GHz Intel E8400 CPU machine.

coverage	h_{LA}	$h_{\text{LM-CUT}}$	$h^+_{\text{LM-CUT}}$
airport (50)	30	28	**31**
blocks (35)	27	**28**	27
depot (22)	7	7	7
driverlog (20)	**14**	13	**14**
elevators-opt08-strips (30)	17	**22**	18
freecell (80)	**58**	15	13
grid (5)	**3**	2	2
gripper (20)	**7**	**7**	6
logistics00 (28)	**21**	20	17
logistics98 (35)	6	6	6
miconic (150)	**142**	141	140
mprime (35)	21	**24**	**24**
mystery (30)	15	**17**	**17**
openstacks-opt08-strips (30)	18	**20**	17
parcprinter-08-strips (30)	15	18	**21**
pathways (30)	4	**5**	**5**
pegsol-08-strips (30)	27	**28**	27
pipesworld-notankage (50)	**18**	17	17
pipesworld-tankage (50)	**13**	12	9
psr-small (50)	**49**	**49**	48
rovers (40)	**8**	7	7
satellite (36)	7	7	**9**
scanalyzer-08-strips (30)	9	**15**	13
schedule (150)	**30**	**30**	27
sokoban-opt08-strips (30)	25	**30**	25
storage (30)	15	15	15
tpp (30)	6	6	6
transport-opt08-strips (30)	**12**	11	11
trucks-strips (30)	9	**10**	9
woodworking-opt08-strips (30)	13	**16**	14
zenotravel (20)	10	**13**	12
SUM	**656**	639	614

Table B.2: Individual performance of h_{LA}, $h_{\text{LM-CUT}}$, $h^+_{\text{LM-CUT}}$ in terms of coverage.

do well overall, while the least informed heuristic (h_{LA}) solved the most tasks in total. However, when looking at the results for individual domains, the best heuristic to use varies, indicating that indeed, combining different heuristics could be of practical use.

We now turn our attention to the empirical results for the combination of every possible subset of two or more heuristics. For each such subset, we will present two tables. The tables marked as (a) list the number of tasks solved in each domain by selective max (sel_h), regular maximum ($\max_h$), and random choice of heuristic at each state (rnd_h) after 30 minutes. This is the accepted measure for performance of optimal planners in the International Planning Competition. The tables marked as (b) measure how informative each combination method is. Since $\max_h$ is the most informative heuristic possible (when combining the given base heuristics), we look at informativeness relative to $\max_h$. We evaluate each planner's informativeness on each task as the number of states expanded by that planner, divided by the number of states expanded by $\max_h$. The table gives the average of this

ratio for each domain, where the average is over the tasks solved by all three combination methods. The number of tasks solved by all planners is listed in parentheses next to each domain. The final row gives the average expansion ratio over all tasks. Table B.3 presents this data for the combination of h_{LA} and $h_{\text{LM-CUT}}$, Table B.4 for the combination of h_{LA} and $h^+_{\text{LM-CUT}}$, Table B.5 for the combination of $h_{\text{LM-CUT}}$ and $h^+_{\text{LM-CUT}}$, and Table B.6 for the combination of all three heuristics.

The results in Tables B.3, B.4, B.5, B.6 clearly demonstrate that when more than one heuristic is in use, selective max is always better than regular maximum or random choice. Furthermore, the poor performance of rnd_h (both in terms of coverage and informativeness) demonstrates that the decision rule and the classifier used in selective max are an important part of its success, and that just computing one heuristic at each state randomly is not enough, to say the least.

Compared to individual heuristics, the selective max combination of h_{LA} and $h_{\text{LM-CUT}}$ solves more tasks than each of the individual heuristics (and every other combination of every other subset of heuristics). However, when $h^+_{\text{LM-CUT}}$ is one of the heuristics used, selective max does not fare as well. The most likely reason for this is that $h^+_{\text{LM-CUT}}$ is very expensive to compute, and even the time spent computing it during the initial sampling period is significant.

Anytime Profile

The time limit of 30 minutes, while commonly used in the IPC, is an arbitrary one, and the number of tasks solved after 30 minutes does not tell the complete tale. Here, we examine the anytime profile of the different heuristic combination methods, by plotting the number of tasks solved under different timeouts, up to a timeout of 30 minutes.

Figure B.1 shows this plot for the three combination methods when all three heuristics are used. As the figure shows, the advantage of sel_h over the other two combination methods is even bigger under shorter timeouts. This indicates that the advantage of sel_h over $\max_h$ is even greater than is evident from the results after 30 minutes, and that sel_h is indeed a better way to minimize search time. As the anytime plots for the combinations of two heuristics are very similar, we omit them for the sake of brevity.

coverage	$\max_h$	rnd_h	sel_h
airport (50)	**30**	28	**30**
blocks (35)	28	28	28
depot (22)	7	7	7
driverlog (20)	**14**	13	**14**
elevators-opt08-strips (30)	**22**	17	**22**
freecell (80)	41	16	**49**
grid (5)	2	2	2
gripper (20)	7	7	7
logistics00 (28)	20	20	**21**
logistics98 (35)	6	6	6
miconic (150)	141	141	**142**
mprime (35)	**24**	18	**24**
mystery (30)	**17**	13	**17**
openstacks-opt08-strips (30)	18	18	18
parcprinter-08-strips (30)	**18**	15	**18**
pathways (30)	**5**	4	**5**
pegsol-08-strips (30)	27	27	27
pipesworld-notankage (50)	17	17	17
pipesworld-tankage (50)	**12**	11	**12**
psr-small (50)	49	49	49
rovers (40)	8	8	8
satellite (36)	7	7	**8**
scanalyzer-08-strips (30)	**15**	7	**15**
schedule (150)	30	30	30
sokoban-opt08-strips (30)	29	**30**	29
storage (30)	15	15	15
tpp (30)	6	6	6
transport-opt08-strips (30)	11	11	11
trucks-strips (30)	**10**	9	**10**
woodworking-opt08-strips (30)	16	13	**17**
zenotravel (20)	**13**	10	**13**
SUM	665	603	**677**

expansions	$\max_h$	rnd_h	sel_h
airport (28)	**1.0**	3.2533	21.5966
blocks (28)	**1.0**	2.5867	1.7004
depot (7)	**1.0**	1.9616	1.3431
driverlog (13)	**1.0**	3.3321	1.2695
elevators-opt08-strips (17)	**1.0**	6.6517	1.5185
freecell (16)	**1.0**	896.0264	2.5223
grid (2)	**1.0**	1.5814	1.8562
gripper (7)	**1.0**	1.0211	**1.0**
logistics00 (20)	**1.0**	1.0002	1.0008
logistics98 (6)	**1.0**	3.4584	1.0825
miconic (141)	1.0	1.0	1.0
mprime (18)	**1.0**	56.4453	2.66
mystery (15)	**1.0**	43.6834	1.5684
openstacks-opt08-strips (18)	**1.0**	1.0357	1.1641
parcprinter-08-strips (15)	**1.0**	132.6402	1.0002
pathways (4)	**1.0**	37.3146	**1.0**
pegsol-08-strips (27)	**1.0**	1.8465	1.0055
pipesworld-notankage (17)	**1.0**	2.3422	1.4337
pipesworld-tankage (11)	**1.0**	1.998	1.1322
psr-small (49)	**1.0**	1.1339	1.1689
rovers (8)	**1.0**	1.9858	1.1093
satellite (7)	**1.0**	2.8051	1.0964
scanalyzer-08-strips (7)	**1.0**	14302.4949	1.1521
schedule (30)	**1.0**	1.0531	1.2054
sokoban-opt08-strips (29)	**1.0**	1.3773	1.0372
storage (15)	**1.0**	1.4903	1.6358
tpp (6)	**1.0**	1.6193	**1.0**
transport-opt08-strips (11)	**1.0**	4.8823	1.6152
trucks-strips (9)	**1.0**	16.6175	1.0119
woodworking-opt08-strips (13)	**1.0**	21.9804	1.4952
zenotravel (10)	**1.0**	7.8704	3.7306
AVERAGE	**1.0**	502.0803	2.0681

Table B.3: h_{LA} / $h_{\text{LM-CUT}}$: Performance summary in terms of coverage (top) and expanded nodes measure, relative to $\max_h$ (bottom).

coverage	$\max_h$	rnd_h	sel_h
airport (50)	**31**	28	30
blocks (35)	**27**	26	26
depot (22)	**7**	6	**7**
driverlog (20)	**14**	13	13
elevators-opt08-strips (30)	**18**	13	16
freecell (80)	31	15	**41**
grid (5)	2	2	2
gripper (20)	5	6	**7**
logistics00 (28)	16	20	**21**
logistics98 (35)	**6**	5	**6**
miconic (150)	140	140	**142**
mprime (35)	**24**	16	**24**
mystery (30)	**17**	12	**17**
openstacks-opt08-strips (30)	16	**17**	**17**
parcprinter-08-strips (30)	21	12	**22**
pathways (30)	**5**	4	**5**
pegsol-08-strips (30)	27	27	27
pipesworld-notankage (50)	**17**	15	**17**
pipesworld-tankage (50)	**9**	8	**9**
psr-small (50)	48	48	**49**
rovers (40)	7	7	**8**
satellite (36)	9	7	**10**
scanalyzer-08-strips (30)	**13**	6	**13**
schedule (150)	27	27	**30**
sokoban-opt08-strips (30)	23	**25**	24
storage (30)	15	15	15
tpp (30)	6	6	6
transport-opt08-strips (30)	11	11	11
trucks-strips (30)	**9**	7	**9**
woodworking-opt08-strips (30)	**15**	11	**15**
zenotravel (20)	**12**	9	**12**
SUM	628	564	**651**

expansions	$\max_h$	rnd_h	sel_h
airport (28)	**1.0**	2.0602	26.2917
blocks (26)	**1.0**	2.4566	5.9109
depot (6)	**1.0**	6.3466	10.005
driverlog (13)	**1.0**	4.3793	3.0043
elevators-opt08-strips (13)	**1.0**	7.6231	11.0712
freecell (15)	**1.0**	192.2481	2.4737
grid (2)	**1.0**	2.1541	5.4812
gripper (5)	1.0	1.0	1.0
logistics00 (16)	1.0	1.0	1.0
logistics98 (5)	**1.0**	2.3065	4.4865
miconic (140)	1.0	1.0	1.0
mprime (16)	**1.0**	22.6245	3.0882
mystery (13)	**1.0**	56.5774	11.1731
openstacks-opt08-strips (16)	**1.0**	1.0338	1.098
parcprinter-08-strips (12)	**1.0**	437.3136	1.2846
pathways (4)	**1.0**	37.545	**1.0**
pegsol-08-strips (27)	**1.0**	2.2176	1.0039
pipesworld-notankage (15)	**1.0**	2.8387	8.8182
pipesworld-tankage (8)	**1.0**	1.6285	2.0812
psr-small (48)	**1.0**	1.1726	1.4059
rovers (7)	**1.0**	2.5532	1.6267
satellite (7)	**1.0**	24.956	7.4651
scanalyzer-08-strips (6)	**1.0**	104.7363	1.4776
schedule (27)	**1.0**	1.0219	1.1224
sokoban-opt08-strips (23)	**1.0**	1.3792	1.0137
storage (15)	**1.0**	1.5795	2.4133
tpp (6)	**1.0**	3.1446	1.2398
transport-opt08-strips (11)	**1.0**	6.0627	3.4795
trucks-strips (7)	**1.0**	17.3822	1.014
woodworking-opt08-strips (11)	**1.0**	14.291	4.4421
zenotravel (9)	**1.0**	7.1326	7.5156
AVERAGE	**1.0**	31.2828	4.3706

Table B.4: h_{LA} / $h^+_{\text{LM-CUT}}$: Performance summary in terms of coverage (top) and expanded nodes measure, relative to $\max_h$ (bottom).

coverage	$\max_h$	rnd_h	sel_h
airport (50)	**31**	27	28
blocks (35)	27	**28**	**28**
depot (22)	7	7	7
driverlog (20)	**14**	13	**14**
elevators-opt08-strips (30)	18	18	**21**
freecell (80)	**13**	12	**13**
grid (5)	2	2	2
gripper (20)	6	6	**7**
logistics00 (28)	16	**20**	**20**
logistics98 (35)	6	6	6
miconic (150)	140	140	**141**
mprime (35)	**24**	21	**24**
mystery (30)	**17**	15	16
openstacks-opt08-strips (30)	17	**19**	**19**
parcprinter-08-strips (30)	**21**	18	20
pathways (30)	5	5	5
pegsol-08-strips (30)	27	27	27
pipesworld-notankage (50)	**17**	16	**17**
pipesworld-tankage (50)	**9**	8	**9**
psr-small (50)	48	48	**49**
rovers (40)	7	7	7
satellite (36)	**9**	7	8
scanalyzer-08-strips (30)	13	13	**15**
schedule (150)	27	27	**30**
sokoban-opt08-strips (30)	25	25	25
storage (30)	15	15	15
tpp (30)	6	6	6
transport-opt08-strips (30)	11	11	11
trucks-strips (30)	9	9	**10**
woodworking-opt08-strips (30)	15	14	**18**
zenotravel (20)	12	12	12
SUM	614	602	**630**

expansions	$\max_h$	rnd_h	sel_h
airport (26)	**1.0**	1.1904	1.4445
blocks (27)	**1.0**	1.0099	1.0257
depot (7)	**1.0**	6.0425	1.2722
driverlog (13)	**1.0**	1.4085	1.5944
elevators-opt08-strips (18)	**1.0**	1.407	1.7418
freecell (12)	**1.0**	4.6317	1.308
grid (2)	**1.0**	2.1345	1.2952
gripper (6)	**1.0**	1.0265	1.0507
logistics00 (16)	1.0	1.0	1.0
logistics98 (6)	**1.0**	1.0661	1.0594
miconic (140)	1.0	1.0	1.0
mprime (21)	**1.0**	4.4253	1.27
mystery (16)	**1.0**	2.5598	1.4236
openstacks-opt08-strips (17)	1.0	1.0	1.0
parcprinter-08-strips (17)	**1.0**	13.1052	**1.0**
pathways (5)	**1.0**	1.1554	1.3181
pegsol-08-strips (27)	**1.0**	1.164	1.2025
pipesworld-notankage (16)	**1.0**	5.9653	1.4612
pipesworld-tankage (8)	**1.0**	2.2528	1.7729
psr-small (48)	**1.0**	1.0175	1.0403
rovers (7)	**1.0**	1.3125	1.3485
satellite (7)	**1.0**	4.5871	1.6311
scanalyzer-08-strips (13)	**1.0**	1.1927	1.174
schedule (27)	1.0	1.0	1.0
sokoban-opt08-strips (25)	**1.0**	1.0149	1.0254
storage (15)	**1.0**	1.0291	1.0801
tpp (6)	**1.0**	1.2044	2.5831
transport-opt08-strips (11)	**1.0**	1.1567	1.3079
trucks-strips (9)	**1.0**	1.1431	1.2694
woodworking-opt08-strips (13)	**1.0**	1.3942	1.1584
zenotravel (12)	**1.0**	1.4235	1.2424
AVERAGE	**1.0**	2.291	1.2936

Table B.5: $h_{\text{LM-CUT}}$ / $h^+_{\text{LM-CUT}}$: Performance summary in terms of coverage (top) and expanded nodes measure, relative to $\max_h$ (bottom).

coverage	$\max_h$	rnd_h	sel_h
airport (50)	**31**	26	30
blocks (35)	27	27	**28**
depot (22)	7	7	7
driverlog (20)	**14**	13	13
elevators-opt08-strips (30)	18	14	**21**
freecell (80)	31	15	**33**
grid (5)	2	2	2
gripper (20)	5	6	**7**
logistics00 (28)	16	**20**	**20**
logistics98 (35)	**6**	5	**6**
miconic (150)	140	140	**142**
mprime (35)	**24**	18	23
mystery (30)	**17**	12	**17**
openstacks-opt08-strips (30)	16	**18**	16
parcprinter-08-strips (30)	**21**	13	19
pathways (30)	**5**	4	**5**
pegsol-08-strips (30)	27	27	27
pipesworld-notankage (50)	**17**	15	**17**
pipesworld-tankage (50)	9	9	**11**
psr-small (50)	48	48	**49**
rovers (40)	7	7	**8**
satellite (36)	**9**	7	8
scanalyzer-08-strips (30)	13	6	**15**
schedule (150)	27	28	**30**
sokoban-opt08-strips (30)	23	**27**	26
storage (30)	15	15	15
tpp (30)	6	6	6
transport-opt08-strips (30)	11	11	11
trucks-strips (30)	9	9	**10**
woodworking-opt08-strips (30)	**15**	14	**15**
zenotravel (20)	**12**	9	**12**
SUM	628	578	**649**

expansions	$\max_h$	rnd_h	sel_h
airport (26)	**1.0**	1.3662	16.8435
blocks (27)	**1.0**	1.6688	1.7768
depot (7)	**1.0**	6.7141	5.9097
driverlog (13)	**1.0**	3.1458	2.3558
elevators-opt08-strips (14)	**1.0**	5.0138	3.1423
freecell (15)	**1.0**	721.2053	18.6437
grid (2)	**1.0**	2.3936	5.7832
gripper (5)	**1.0**	1.0195	1.0007
logistics00 (16)	1.0	1.0	1.0
logistics98 (5)	**1.0**	1.8846	1.6551
miconic (140)	1.0	1.0	1.0
mprime (18)	**1.0**	15.3643	6.5826
mystery (13)	**1.0**	26.294	2.003
openstacks-opt08-strips (16)	**1.0**	1.0261	1.079
parcprinter-08-strips (13)	**1.0**	1069.3099	**1.0**
pathways (4)	**1.0**	13.5048	1.0017
pegsol-08-strips (27)	**1.0**	1.7488	1.2814
pipesworld-notankage (15)	**1.0**	5.7717	8.8866
pipesworld-tankage (9)	**1.0**	2.9312	2.1266
psr-small (48)	**1.0**	1.1148	1.3725
rovers (7)	**1.0**	2.2181	1.5002
satellite (7)	**1.0**	13.5424	3.0183
scanalyzer-08-strips (6)	**1.0**	41.0034	1.9857
schedule (27)	1.0	**0.9956**	1.1224
sokoban-opt08-strips (23)	**1.0**	1.149	1.2074
storage (15)	**1.0**	1.3521	1.6715
tpp (6)	**1.0**	3.9019	1.7979
transport-opt08-strips (11)	**1.0**	3.2206	2.2665
trucks-strips (9)	**1.0**	7.5212	1.3416
woodworking-opt08-strips (11)	**1.0**	5.5086	2.025
zenotravel (9)	**1.0**	5.3099	3.8208
AVERAGE	**1.0**	63.5226	3.4259

Table B.6: h^{LA} / $h^{\text{LM-CUT}}$ / $h^{+}_{\text{LM-CUT}}$: Performance summary in terms of coverage (top) and expanded nodes measure, relative to $\max_h$ (bottom).

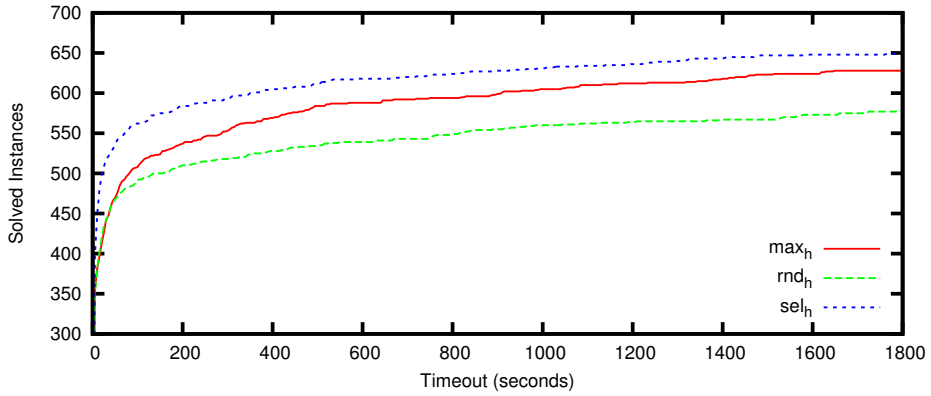


Figure B.1: h_{LA} / $h_{\text{LM-CUT}}$ / $h^+_{\text{LM-CUT}}$: Anytime performance in terms of coverage.

B.2 Impact of Parameter Settings

For the second part of our empirical evaluation, we evaluate the impact of different parameter settings on selective max. We focus on the best choice of heuristics for selective max — combining h_{LA} and $h_{\text{LM-CUT}}$. For the experiments described in this section, we take the default configuration, which is described in Table B.1, and evaluate the impact of setting each parameter to several different values. Since this results in over 20 different configurations, we ran this part of the empirical evaluation on 8 selected domains, which were chosen from the 31 domains in the first part of the evaluation because they had the highest variance in the number of tasks solved across different configurations. In each subsection below we focus on one parameter of selective max, and present a table with the number of tasks solved in each of our 8 chosen domains under different values of that parameter.

Hyper-parameter α

The hyper-parameter α controls the tradeoff between computation time and heuristic accuracy. Setting $\alpha = 0$ sets the threshold τ to 0, essentially telling the decision rule to always choose the more informed heuristic (that is, choose h_2 iff it is estimated that $h_2(s) > h_1(s)$). Increasing α increases the threshold, forcing the decision rule to choose the more informed heuristic h_2 only if its value is much higher than that of h_1.

As the results in Table B.7 show, selective max is fairly robust to values

coverage	$\text{sel}_h^{\alpha=0.1}$	$\text{sel}_h^{\alpha=0.5}$	$\text{sel}_h^{\alpha=1}$	$\text{sel}_h^{\alpha=2}$	$\text{sel}_h^{\alpha=5}$
airport (50)	30	30	30	30	30
freecell (80)	49	49	49	49	49
logistics00 (28)	21	21	21	21	21
mprime (35)	**24**	**24**	**24**	22	21
mystery (30)	**17**	**17**	**17**	**17**	15
pipesworld-tankage (50)	12	12	12	12	**13**
satellite (36)	**8**	**8**	**8**	7	7
zenotravel (20)	**13**	**13**	**13**	12	10
SUM	**174**	**174**	**174**	170	166

Table B.7: Hyper-parameter α.

of α. However, if α is too large, we do see performance degradation, which indicates that our default value of $\alpha = 1$ is a fairly good choice.

Confidence Threshold ρ

coverage	$\text{sel}_h^{\rho=0.51}$	$\text{sel}_h^{\rho=0.6}$	$\text{sel}_h^{\rho=0.7}$	$\text{sel}_h^{\rho=0.8}$	$\text{sel}_h^{\rho=0.9}$	$\text{sel}_h^{\rho=0.99}$
airport (50)	30	30	30	30	30	30
freecell (80)	48	**49**	**49**	**49**	**49**	**49**
logistics00 (28)	21	21	21	21	21	21
mprime (35)	24	24	24	24	24	24
mystery (30)	17	17	17	17	17	17
pipesworld-tankage (50)	12	12	12	12	12	12
satellite (36)	8	8	8	8	8	8
zenotravel (20)	13	13	13	13	13	13
SUM	173	**174**	**174**	**174**	**174**	**174**

Table B.8: Confidence threshold ρ.

The confidence threshold ρ controls the active learning part of selective max. Setting $\rho = 0.5$ turns off active learning completely (as the chosen heuristic always has confidence at least 0.5), while setting $\rho = 1$ would mean using active learning almost always, essentially reducing selective max to regular point-wise maximization. The results in Table B.8 indicate that selective max is also robust to values of ρ, unless it is set to a very low value.

Initial Sample Size t

coverage	$\text{sel}_h^{t=10}$	$\text{sel}_h^{t=100}$	$\text{sel}_h^{t=1000}$
airport (50)	30	30	30
freecell (80)	47	**49**	46
logistics00 (28)	21	21	21
mprime (35)	24	24	24
mystery (30)	17	17	17
pipesworld-tankage (50)	12	12	12
satellite (36)	8	8	8
zenotravel (20)	13	13	13
SUM	172	**174**	171

Table B.9: Initial Sample Size t

The initial sample size t is an important parameter, not just because it is used to train the initial classifier (before any active learning is done), but also because it is the only source of estimates for branching factor and heuristic computations times, thus controlling the threshold τ. Increasing t increases the accuracy of the initial classifier and of the estimates for branching factor and heuristic computations times, at the cost of more preprocessing time.

As the results in Table B.9 show, our default value of $t = 100$ is the best (of the 3 values we tried), although selective max is still fairly robust to the choice of parameter.

Sampling Method

coverage	sel_h^{PDB}	sel_h^{P}	sel_h^{UP}
airport (50)	30	30	30
freecell (80)	49	53	**55**
logistics00 (28)	21	21	21
mprime (35)	24	24	24
mystery (30)	17	17	17
pipesworld-tankage (50)	12	12	12
satellite (36)	8	8	8
zenotravel (20)	13	13	13
SUM	174	178	**180**

Table B.10: Sampling method.

The sampling method used is also an important parameter which affects the initial sample, and thus the accuracy of the threshold τ and of the initial classifier. The default sampling method (in this version of selective max) is the sampling method of Haslum et al. (2007). This sampling method is based on unbiased random walks (probes), but only adds the last state reached in each probe. The depth of each probe is distributed binomially around the estimated goal depth. We refer to the planner using this sampling method as (sel_h^{PDB}).

The default (and only) method in previous versions was the one described in Section 5.3.1, with stochastic probes biased towards states with low heuristic values (sel_h^{P}). Another version we compared to used a similar sampling method, except that the random walks were unbiased, and the successor to move to was chosen uniformly (sel_h^{UP}).

As the results in Table B.10 demonstrate, the choice of sampling method can have a significant effect. However, since the effect is only evident in the FREECELL domain, we believe this effect is more due to the time spent in heuristic computation during the initial sample, rather than due to the quality of the sample.

Classifier

coverage	sel_h^{NB}	sel_h^{AODE}	sel_h^{ITI}	sel_h^{3NN}	sel_h^{5NN}
airport (50)	**30**	25	**30**	**30**	28
freecell (80)	**49**	**49**	34	35	46
logistics00 (28)	**21**	20	20	20	20
mprime (35)	**24**	**24**	**24**	**24**	23
mystery (30)	17	17	17	17	17
pipesworld-tankage (50)	**12**	**12**	**12**	**12**	10
satellite (36)	**8**	**8**	7	7	6
zenotravel (20)	**13**	**13**	12	**13**	11
SUM	**174**	168	156	158	161

Table B.11: Classifier.

Finally, the choice of classifier is also very important. Our default choice is the Naive Bayes classifier (Mitchell, 1997), which combines very fast learning and classification (sel_h^{NB}). A more sophisticated variant of Naive Bayes called AODE (Webb et al., 2005) is also considered here (sel_h^{AODE}). AODE is more accurate than Naive Bayes, at the cost of higher classification and learning times (as well as increased memory overhead). Another possible choice is using incremental decision trees (Utgoff et al., 1997), which offer even faster classification, but more expensive learning when the tree structure needs to be changed (sel_h^{ITI}). We also consider kNN classifiers (Cover and Hart, 1967), which offer faster learning than Naive Bayes, but usually more expensive classification, especially as k grows larger (sel_h^{kNN}, for $k = 3, 5$).

As the results in Table B.11 show, Naive Bayes is indeed the best classifier to use with selective max, although AODE performs quite well. As expected, kNN does not do as well, since the classifier is used mostly for classification, which is expensive for kNN. However, the increased accuracy of $k = 5$ seems to pay off against the faster classification when $k = 3$.

Bibliography

Christer Bäckström and Inger Klein. Planning in polynomial time: the SAS-PUBS class. *Computational Intelligence*, 7(3):181–197, 1991.

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

A. Bagchi and A. Mahanti. Search algorithms under different kinds of heuristics-a comparative study. *Journal of the ACM*, 30(1):1–21, 1983.

Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI 1997)*, pages 203–208. AAAI Press, 1997.

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

Blai Bonet and Malte Helmert. Strengthening landmark heuristics via hitting sets. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 329–334. IOS Press, 2010.

Blai Bonet, Gábor Loerincs, and Héctor 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. AAAI Press, 1997.

Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24:581–621, 2005.

Olivier Buffet and Jörg Hoffmann. All that glitters is not gold: Using landmarks for reward shaping in FPG. In *ICAPS 2010 Workshop on Planning and Scheduling Under Uncertainty*, 2010.

Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, chapter 16, pages 457–474. 2003.

Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.

Yixin Chen and Guohui Yao. Completeness and optimality preserving reduction for planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1659–1664, 2009.

Andrew Coles and Amanda Smith. Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research*, 28:119–156, 2007.

Andrew. I. Coles and Amanda. J. Smith. Upwards: The role of analysis in cost optimal SAS+ planning. In *Sixth International Planning Competition (IPC-6): planner abstracts*, 2008.

Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21 – 27, 1967.

Tomás de la Rosa, Sergio Jiménez, and Daniel Borrajo. Learning relational decision trees for guiding heuristic planning. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, editors, *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, pages 60–67. AAAI Press, 2008.

Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536, 1985.

Carmel Domshlak, Malte Helmert, Erez Karpas, Emil Keyder, and Silvia Richter. Landmarks in optimal planning. Working paper, 2012.

Stefan Edelkamp. Planning with pattern databases. In Amedeo Cesta and Daniel Borrajo, editors, *Pre-proceedings of the Sixth European Conference on Planning (ECP 2001)*, pages 13–24, Toledo, Spain, 2001.

Stefan Edelkamp and Malte Helmert. The model checking integrated planning system (MIPS). *AI Magazine*, 22(3):67–71, 2001.

Stefan Edelkamp and Stefan Schrödl. *Heuristic Search: Theory and Applications*. Morgan Kaufmann, 2011.

Alan Fern. Speedup learning. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, pages 907–911. Springer, 2010.

Alan Fern, Roni Khardon, and Prasad Tadepalli. The first learning track of the international planning competition. *Machine Learning*, 84(1-2): 81–107, 2011.

Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

Lev Finkelstein and Shaul Markovitch. A selective macro-learning algorithm and its application to the NxN sliding-tile puzzle. *Journal of Artificial Intelligence Research*, 8:223–263, 1998.

Maria Fox and Derek Long. Extending the exploitation of symmetries in planning. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, pages 83–91. AAAI Press, 2002.

Ángel García-Olaya, Sergio Jiménez, and Carlos Linares López. The 2011 international planning competition. Technical report, Universidad Carlos III de Madrid, 2011. http://hdl.handle.net/10016/11710.

Hector Geffner. The model-based approach to autonomous behavior: A personal view. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 1709–1712. AAAI Press, 2010.

David Gelperin. On the optimality of A^*. *Artificial Intelligence*, 8:69–76, 1977.

Larry R. Harris. The heuristic search under conditions of error. *Artificial Intelligence*, 5(3):217–234, 1974.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Patrik Haslum and Héctor Geffner. Admissible heuristics for optimal planning. In Steve Chien, Subbarao Kambhampati, and Craig A. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*, pages 140–149. AAAI Press, 2000.

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, pages 1007–1012. AAAI Press, 2007.

Malte Helmert. *Understanding Planning Tasks – Domain Complexity and Heuristic Decomposition*, volume 4929 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2008.

Malte Helmert. personal communication, 2009a.

Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173:503–535, 2009b.

Malte Helmert. A planning heuristic based on causal graph analysis. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 161–170. AAAI Press, 2004.

Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 162–169. AAAI Press, 2009.

Malte Helmert and Héctor Geffner. Unifying the causal graph and additive heuristics. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, editors, *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, pages 140–147. AAAI Press, 2008.

Malte Helmert and Gabriele Röger. How good is almost perfect? In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 944–949. AAAI Press, 2008.

Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In Mark Boddy, Maria Fox, and Sylvie Thiébaux, editors, *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, pages 176–183. AAAI Press, 2007.

Malte Helmert, Gabriele Röger, and Erez Karpas. Fast Downward Stone Soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, pages 28–35, 2011.

Jörg Hoffmann. Local search topology in planning benchmarks: A theoretical analysis. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, pages 92–100. AAAI Press, 2002.

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

Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

Erez Karpas and Carmel Domshlak. Living on the edge: Safe search with unsafe heuristics. In *ICAPS 2011 Workshop on Heuristics for Domain-Independent Planning*, 2011.

Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1728–1733, 2009.

Michael Katz and Carmel Domshlak. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, 174(12–13):767–798, 2010a.

Michael Katz and Carmel Domshlak. Implicit abstraction heuristics. *Journal of Artificial Intelligence Research*, 39:51–126, 2010b.

Henry Kautz and Bart Selman. Planning as satisfiability. In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 1992)*, pages 359–363. John Wiley and Sons, 1992.

Emil Keyder, Silvia Richter, and Malte Helmert. Sound and complete landmarks for and/or graphs. In Helder Coelho, Rudi Studer, and Michael

Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 335–340. IOS Press, 2010.

Peter Kissmann and Stefan Edelkamp. Improving cost-optimal domain-independent symbolic planning. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, pages 992–997. AAAI Press, 2011.

Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

Nir Lipovetzky and Hector Geffner. Searching for plans with carefully designed probes. In Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp, and Malte Helmert, editors, *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)*, pages 154–161. AAAI Press, 2011.

João P. Marques-Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1996)*, pages 220–227, 1996.

Alberto Martelli. On the complexity of admissible search algorithms. *Artificial Intelligence*, 8(1):1–13, 1977.

David Mcallester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI 1991)*, pages 634–639. AAAI Press/MIT Press, 1991.

László Mérő. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence*, 23(1):13–27, 1984.

Steven Minton. *Machine Learning Methods for Planning*. Morgan Kaufmann Publishers Inc., 1994.

Tom Michael Mitchell. *Machine Learning*. WCB/McGraw-Hill, 1997.

Srinivas Nedunuri, William R. Cook, and Douglas R. Smith. Cost-based learning for planning. In *ICAPS 2011 Workshop on Planning and Learning*, pages 68–75, 2011.

Raz Nissim, Jörg Hoffmann, and Malte Helmert. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Toby Walsh, editor, *Proceedings of the*

22nd International Joint Conference on Artificial Intelligence (IJCAI'11), pages 1983–1990. AAAI Press/IJCAI, 2011.

Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

Edwin P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR 1989)*, pages 324–332. Morgan Kaufmann, 1989.

J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR 1992)*, pages 103–114. Morgan Kaufmann, 1992.

Nir Pochter, Aviv Zohar, and Jeffrey S. Rosenschein. Exploiting problem symmetries in state-based planners. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, pages 1004–1009. AAAI Press, 2011.

Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3):193–204, 1970.

Julie Porteous and Stephen Cresswell. Extending landmarks analysis to reason about resources and repetition. In *Proceedings of the 21st Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG '02)*, pages 45–54, 2002.

Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.

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. AAAI Press, 2008.

Jussi Rintanen. Symmetry reduction for SAT representations of transition systems. In Enrico Giunchiglia, Nicola Muscettola, and Dana Nau, editors, *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003)*, pages 32–40. AAAI Press, 2003.

Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170 (12–13):1031–1080, 2006.

Stuart Russell and Peter Norvig. *Artificial Intelligence — A Modern Approach*. Prentice Hall, 2010.

Thomas Schiex and Grard Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 3:48–55, 1993.

Austin Tate. Generating project networks. In Raj Reddy, editor, *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI 1977)*, pages 888–893. William Kaufmann, 1977.

Paul E. Utgoff, Neil C. Berkman, and Jeffery A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1): 5–44, 1997.

Vincent Vidal and Héctor Geffner. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence*, 170(3):298–335, 2006.

Geoffrey I. Webb, Janice R. Boughton, and Zhihai Wang. Not so naive Bayes: Aggregating one-dependence estimators. *Machine Learning*, 58 (1):5–24, 2005.

Sungwook Yoon, Alan Fern, and Robert Givan. Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, 9: 683–718, 2008.

Zhifu Zhang, Nathan R. Sturtevant, Robert Holte, Jonathan Schaeffer, and Ariel Felner. A* search with inconsistent heuristics. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 634–639, 2009.

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

Terry Zimmerman and Subbarao Kambhampati. Learning-assisted automated planning: looking back, taking stock, going forward. *AI Magazine*, 24:73–96, 2003.

היוריסטיקות לא קלאסיות עבור תכנון קלאסי

ארז כרפס

היוריסטיקות לא קלאסיות עבור תכנון קלאסי

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
דוקטור לפילוסופיה

ארז כרפס

הוגש לסנט הטכניון - מכון טכנולוגי לישראל
2012 אדר תשע"ב חיפה מרץ

המחקר נעשה בהנחיית פרופ' כרמל דומשלק ופרופ' שאול מרקוביץ' בפקולטה
להנדסת תעשייה וניהול

הנני מודה לטכניון על התמיכה הכלכלית הנדיבה בהשתלמותי

תקציר

תכנון אוטומטי מהווה את הגישה מבוססת המודל להתנהגות אוטונומית של סוכנים, והוא אחד התחומים החשובים והמוכרים ביותר של בינה מלאכותית. תכנון אוטומטי עוסק בהשגת מטרה מסוימת ממצב התחלתי נתון, באמצעות שימוש באוסף פעולות נתון. ישנם מודלים אפשריים שונים, ביניהם מודלים מורכבים שעוסקים בידע לא ודאי לגבי מצב העולם, פעולות עם אפקטים לא דטרמיניסטיים, תהליכים טמפורליים, ועוד. בתכנון קלאסי, אנו עוסקים באחד המודלים הפשוטים ביותר – הכולל ידע ודאי לגבי מצבי העולם, פעולות דטרמיניסטיות לחלוטין, והפעלת פעולות סדרתית. למרות פשטות המודל, אפילו הכרעה אם יש לבעיית תכנון נתונה פתרון היא בעייה קשה חישובית.

בתכנון אופטימלי, המטרה אינה רק למצוא פתרון כלשהו, אלא למצוא אחד הפתרונות הטובים (כלומר, זולים) ביותר. מצד שני, בתכנון מספק, גם פתרון שאינו אופטימלי הוא פתרון קביל. באופן כללי, תכון אופטימלי ותכנון מספק קשים חישובית באותה המידה. למרות זאת, עבור בעיות תכנון רבות, תכנון מספק ניתן לפתרון בצורה יעילה, אך תכנון אופטימלי הוא קשה חישובית.

אנו מתמקדים בתכנון אופטימלי. בשנים האחרונות, התקדמות אדירה בתכנון אופטימלי נבעה מטכניקות חדשות לייצור אוטומטי של היוריסטיקות קבילות, המגיעות ממשפחות שונות של היוריסטיקות. למרות הגיוון הרב בהיוריסטיקות אלה, לכולן יש מכנה משותף – כולן מה שאנו מכנים קלאסיות, היוריסטיקות שמעריכות את המרחק ממצב מסוים למטרה, על פי תכונות של המצב בלבד.

בעבודה זו, נעסוק בהיוריסטיקות לא-קלאסיות – היורסטיקות שמשתמשות במידע נוסף שנאסף תוף כדי חיפוש, ולא בתכונות של המצב שהן מעריכות בלבד. למרבה הפתעתנו, הספרות שעוסקת בחיפוש היוריסטי לא מציעה הגדרה מתמטית שמאפשרת לתאר היוריסטיקות לא-קלאסיות באופן פורמאלי. לכן, לאחר סקירה קצרה של הרקע ההכרחי בפרק 2, התרומה הראשונה שלנו, המוצגת בפרק 3, היא מודל פורמאלי, המאפשר לנו להגדיר היוריסטיקות לא-קלאסיות בצורה מתמטית. כמו כן, אנו מתארים חלוקה של היוריסטיקות לא קלאסיות לפי מספר מימדים, ומציגים אלגוריתמי חיפוש המותאמים להיוריסטיקות לא-קלאסיות בצורה יותר טובה מאלגוריתמי חיפוש קיימים.

על מנת להראות שהיוריסטיקות לא-קלאסיות אינן רק אפשרות תיאורטית מעניינת, אלא גם מהווים דרך טבעית לנצל מידע שנאסף תוך כדי חיפוש, אנו מציגים היוריסטיקות לא-קלאסיות ממשפחות שונות. בפרק 4 נציג היוריסטיקות המבוססות על סימני-דרך, המנצלות מידע מהמסלול הידוע (או המסלולים הידועים) אל המצב שהן מעריכות. סימן-דרך הוא נוסחה לוגית מסוימת, שצריכה להתקיים בנקודה כלשהי לאורך כל פתרון של בעיית התכנון. ניתן לחשב מראש קבוצה של סימני-דרך שחייבים להתקיים, ולהשתמש בסימני הדרך שלא "הושגו" על מנת לקבל היורסטיקה קבילה. על מנת לדעת אילו סימני-דרך כבר הושגו, לא מספיק להסתכל על המצב הנוכחי, אלא יש צורך להסתכל על המסלול בעזרתו הגענו אל מצב זה. אנו מראים שעל ידי הצלבת מידע ממספר מסלולים שונים, ניתן לקבל עוד יותר מידע, ולהסיק שלפעמים, למרות שסימן-דרך מסוים הושג, יש צורך להשיג אותו שוב. כמו כן, אנו מראים טכניקת היסק חדשה, המסתכלת על מסלול מסוים, ומסיקה אילו פעולות צריכות להופיע בהמשך המסלול, על מנת שמסלול זה יהיה מועמד להיות התחלה של פתרון אופטימלי. על סמך טכניקת היסק זו אנו מקבלים היוריסטיקה לא קבילה, שבכל זאת מאפשרת לנו לקבל פתרון אופטימלי לבעיות תכנון.

בפרק 5 נציג היוריסטיקות המבוססות על למידה חישובית, המנצלות מידע שנאסף לאורך כל תהליך החיפוש, ולא רק במסלולים המובילים למצב שהיוריסטיקה מעריכה. נתאר מערכת הלומדת תוך כדי חיפוש, ומחליטה איזו היוריסטיקה קבילה כדאי להפעיל בכל מצב, על מנת להקטין את זמן החיפוש. מערכת זו מתבססת על מודל תיאורטי שפיתחנו, על פיו אנו מגדירים כלל החלטה שקובע, בהינתן מידע מלא, שימוש באיזו היוריסטיקה יקטין את זמן החיפוש. כלל החלטה זה מהווה את הקונספט שאותו אנו מנסים ללמוד בעזרת מסווג, המנסה לנבא, בעזרת מידע חלקי שנלמד לאורך החיפוש, מה יאמר כלל ההחלטה בהינתן מידע מלא. בנוסף, אנו מנצלים מידע לגבי רמת הבטחון של המסווג, על מנת להחליט "לשלם" בזמן, וללמוד מדוגמה שנתקלנו בה תוך כדי חיפוש, ולכן המערכת שלנו היא מערכת לומדת אקטיבית.

בפרק 6 אנו מציגים תוצאות של ניסויים אמפיריים, המראים שהיוריסטיקות אלה הן כלים שימושיים בפועל, ולמעשה הובילו לזכייה במקום השני בתחרות התכנון הבין-לאומית האחרונה, ב-2011. לבסוף, אנו מציגים בפרק 7 את מסקנותינו.