UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

AUGUSTO BLAAS CORRÊA

Domain-Dependent Heuristics and Tie-Breakers: Topics in Automated Planning

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. André G. Pereira
Coadvisor: Prof. Dr. Marcus Ritt

Porto Alegre
January 2018

“Ich sage euch: man muss noch Chaos in sich haben,
um einen tanzenden Stern gebären zu können”
— FRIEDRICH NIETZSCHE

ACKNOWLEDGMENTS

I thank you, my dear reader.

ABSTRACT

Automated planning is an important general problem solving technique in Artificial Intelligence (AI). In planning, given a initial state of the world, a goal and a set of actions, we want to find a sequence of these actions leading us to the goal. What makes planning interesting is that we can model several different domains into planning tasks and solve them using a single method (e.g. Sokoban, logistic problems, security verification, etc). In this thesis, we will approach two different aspects of planning.

First, we study domain-dependent heuristics in the context of the airport ground traffic problem and compare them to the state-of-the-art heuristics in literature. We present heuristics that over perform any other known method. Also, we show that simple domain-dependent heuristics can still be superior than the state-of-the-art domain-independent ones. These proposed heuristics resulted in a conference paper published (CORRÊA; PEREIRA; RITT, 2016) previously.

The second part of this work treats about tie-breakers for A* search algorithm. In a more theoretical fashion, we study the current tie-breakers in literature (ASAI; FUKUNAGA, 2016; ASAI; FUKUNAGA, 2017) and show that all these techniques can mislead the search. We propose a new method based on operator (transition) cost adaptation that is proved to be the best possible tie-breaker in the perfect scenario. We also show that the methods proposed can solve more instances – i.e., increase coverage – than the “canonical” methods in literature.

We believe that the analysis on domain-dependent heuristics and the tie-breaking strategies proposed are valuable not only to the automated planning community. Anyone interested in search, general AI topics, general problem solving or even just puzzles and games can benefit from this work.

Keywords: Heuristic search. automated planning. airport control. A*. tie-breaker.

Heurísticas dependentes de domínio e regras de desempate: tópicos planejamento automatizado

RESUMO

Planejamento automatizado é uma importante técnica genérica de solução de problemas em Inteligência Artificial (IA). Em planejamento, dado um estado inicial do mundo, um objetivo e um conjunto de ações, queremos encontrar a sequência destas ações que nos leva para o objetivo. O que torna planejamento interessante é o fato de que podemos modelar diversos domínios em tarefas de planejamento e resolvê-los usando um único método (e.g. Sokoban, problemas de logística, verificação de segurança, etc.). Nesta tese, abordamos dois aspectos diferentes de planejamento.

Primeiro, estudamos heurísticas dependentes de domínio no contexto do problema de tráfego de aeroportos em solo e comparamos estas com as principais heurísticas da literatura. Nós apresentamos heurísticas que superam qualquer outro método conhecido. Mostramos também que simples heurísticas dependentes de domínio ainda podem ser superiores as principais heurísticas independentes de domínio. As heurísticas propostas resultaram em um artigo publicado anteriormente em uma conferência (CORRÊA; PEREIRA; RITT, 2016).

A segunda parte deste trabalho trata sobre regras de desempate em planejamento. De uma maneira mais teórica, estudamos os atuais métodos de desempate na literatura e mostramos que tais técnicas apresentam problemas. Nós propomos um novo método baseado na adaptação de custos que é provado ser o melhor possível. Também demonstramos que tais métodos propostos podem solucionar mais instâncias que os métodos padrões na literatura.

Acreditamos que os resultados apresentados sobre heurísticas independentes de domínio e regras de desempate têm valor não só para a comunidade do planejamento automatizado. Qualquer um interessado em busca, IA em geral, soluções de problemas em geral ou até mesmo apenas em jogos pode se beneficiar deste trabalho.

Palavras-chave: Busca heurística, planejamento automatizado, controle de aeroportos, A* , regras de desempate.

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\mathcal{V}$	Set of finite-domain variables
$\mathcal{O}$	Set of operators
s_0	Initial state (complete variable assignment) of a planning task
s_*	Partial variable assignment indicating the goal fact of a planning task
S_*	Set of states that contain the partial variable assignment s_*
cost	Operator cost function cost : $\mathcal{O} \rightarrow \mathbb{R}^0_+$
Π	Planning task defined by the tuple $\langle \mathcal{V}, \mathcal{O}, s_0, s_*, \text{cost} \rangle$
$\mathcal{S}$	State space induced by a planning task Π defined by the tuple $\langle s_0, S_*, \mathcal{O}, \text{cost} \rangle$; set of reachable states in Π
h	Heuristic function $h : \mathcal{S} \rightarrow \mathbb{R}^0_+ \cup \{\infty\}$
h^*	Perfect heuristic function $h^* : \mathcal{S} \rightarrow \mathbb{R}^0_+ \cup \{\infty\}$
g	Real cost function $g : \mathcal{S} \rightarrow \mathbb{R}^0_+ \cup \{\infty\}$
f	State evaluation function $f : \mathcal{S} \rightarrow \mathbb{R}^0_+ \cup \{\infty\}$
f^*	State evaluation function using the perfect heuristic $f : \mathcal{S} \rightarrow \mathbb{R}^0_+ \cup \{\infty\}$
d	Distance function in number of operators from a state to its closest goal state in an optimal plan $f : \mathcal{S} \rightarrow \mathbb{N} \cup \{\infty\}$
$\mathcal{C}^*$	Cost of an optimal plan

CONTENTS

1 INTRODUCTION

Automated planning is a general problem-solving technique in Artificial Intelligence (AI) where, given an initial state of a problem, we try to find a sequence of operators leading us to a goal. For example, in an airport ground-traffic control problem we may want to find a sequence of movements that planes must perform to reach their desired position given a set of constraints in the airport. A vast number of domains can be seen as planning tasks, such as logistics problems, transportation tasks (e.g. Sokoban), puzzles (e.g. $(n^2 - 1)$-puzzle) and scheduling problems.

The goal of *domain-independent planning* is to create an algorithm that performs well in any given problem – i.e., methods that do not use any prior information related to the task. In this work, we are interested in a specific formalism of domain-independent planning, called classical planning. In this formalism, a single agent executes a sequence of operators, with deterministic, discrete and fully observable effects, until it reaches the goal. One could relax this formalism to obtain other variants of planning, such as partially observable, probabilistic or multi-agent planning. Beyond that, classical planning can be split into two categories: optimal and satisficing planning. While the former tries to find the best solution for a planning task, the latter only tries to find a feasible solution.

Different planning domains can present different complexities, however, classical planning is PSPACE-complete (HELMERT, 2006b). Also, optimal and satisficing planning can present different complexities for a same domain. For instance, the BLOCKSWORLD domain – where we have a table full of numbered blocks and they need to be organized in a specific manner – is in NP-complete for optimal planning and in P for satisficing planning, while the airport domain (mentioned in the first paragraph) is PSPACE-complete for both. This natural hardness of planning forces us to look for methods to reduce our computational effort (HELMERT, 2006b; HELMERT, 2001).

In the last two decades, one of the main techniques to solve planning problems is based on *heuristic search methods* (BONET; GEFFNER, 2001). A *heuristic*, in the sense of this work, is an informed function that tries to estimate the cost from a state of the task to its goal. By performing an informed search (e.g. A* (HART; NILSSON; RAPHAEL, 1968), Greedy Best-first Search (DORAN; MICHIE, 1966)), we try to reduce the effort needed to find a solution for a planning task by reducing the state explosion problem.

This work approaches two different issues in classical planning. First, given the last advancements in planning using heuristic search, we want to solve the question:

are the state-of-the-art heuristics in planning better than simple, almost trivial, domain-dependent heuristics? To do so, we will investigate one of the hardest domains in planning, the airport ground-traffic problem. Second, we address the problem of tie-breakers in heuristic search, more specifically in A* algorithm and ask: *can we find a perfect tie-breaker for A*?*

In the following chapter, we provide the background needed for the rest of this work and, then, we move further to answer the questions stated above. The two main chapters of this thesis are not directly related; they present the results and contributions from two different works, thus they were organized to be self-contained (i.e., each chapter has its own "Related Work" and "Conclusion" subsections). Before doing so, we outline the contributions of this thesis and its relation to previous published works.

1.1 Thesis Outline and Relation to Previous Work

This thesis has two clearly divided parts. The first part is dedicated to domain-dependent heuristics in the airport ground-traffic problem. We show that heuristics based on simple calculated distances from each airplane to its destination can be more informed and present better coverage (i.e., Number of solved instances[1]) than the state-of-the-art domain-independent heuristics present in the Fast-Downward planner (HELMERT, 2006a). This work was previously published as a paper in the BRACIS 2016 (Brazilian Conference on Intelligent Systems) as "*Improved Airport Ground Traffic Control with Domain-Dependent Heuristics*" (CORRÊA; PEREIRA; RITT, 2016). Chapter 3 contains all the results and some fragments from Corrêa, Pereira and Ritt (2016).

The second part refers to the theoretical study of tie-breaking strategies for A*. We present a theoretical method that guarantees optimal expansion for the search under best scenario assumptions. Our theoretical results are empirically tested and we obtain better performance on coverage than the state-of-the-art in literature.

[1]Using *coverage* to express *number of solved instances* in an experiment is common in the classical planning literature and, due to that, we will use them as synonymous in this work.

2 BACKGROUND: PLANNING AND HEURISTIC SEARCH

In this chapter, we introduce the main notations and definitions used in this work. A more experienced reader could skip some sections and subsections, consulting the List of Symbols in case of need. We start by defining the planning formalism (Section 2.1), then moving to definitions about heuristics and common heuristics in planning literature (Section 2.2). Finally, we discuss heuristic search and search algorithms (Section 2.3).

2.1 Planning Formalism

We consider the SAS$^+$ formalism from Bäckström and Nebel (1995) to define planning tasks. SAS$^+$ tasks are based on a set of finite-domain *variables* $\mathcal{V}$. Each variable $v \in \mathcal{V}$ has a finite domain $\text{dom}(v)$ of possible values that can be assigned to it. An assignment $\langle v, d \rangle$ where $v \in \mathcal{V}$ and $d \in \text{dom}(v)$ is a *fact*. If a variable v appears in a fact $\langle v, d \rangle$ we say that it *contains* v (or, v is contained in $\langle v, d \rangle$).

Definition 2.1 (Partial and Complete Assignments)**.** *Let $\mathcal{V}$ be a set of finite-domain variables, a* partial variable assignment *is a set of facts where each variable $v \in \mathcal{V}$ is contained in at most one fact; a* complete variable assignment *is a set of facts where each variable $v \in \mathcal{V}$ is contained in* exactly *one fact.*

A complete variable assignment is also called a *state*. Notice that a partial variable assignment can also be a state, but this is not always the case.

In a nutshell, a *planning task* (also called planning problem or planning instance) can be seen as a set of variables, a initial state composed of facts over these variables, a partial variable assignment indicating the goal and a set of operators (with specific cost each) to modify a state.

Definition 2.2 (Planning Task)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*, cost \rangle$ be a planning task, where $\mathcal{V}$ is a set of variables, $\mathcal{O}$ is a set of operators, s_0 is the initial state, s_* is the goal and cost is a function applicable in each element of $\mathcal{O}$.*

The initial state s_0 is a complete variable assignment and the goal s_* is a partial variable assignment. Any state that contains the partial variable assignment s_* is called a *goal state*.

Each operator $o \in \mathcal{O}$ is defined as a tuple $o = \langle \text{pre}(o), \text{eff}(o) \rangle$, where $\text{pre}(o)$ and $\text{eff}(o)$ are partial variable assignments representing the operator's *precondition* (i.e., a set of facts that must be true in a state s to apply o) and *effect* (i.e., a set facts that will become true if the operator o is applied to s), respectively. An operator o is only *applicable* to a state s if s contains the partial assignment pre(o). The function cost: $\mathcal{O} \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$ maps an operator $o \in \mathcal{O}$ to the cost of applying o to a state. Applying an operator o in a state s will generate the *successor state* $s[o]$. The set of all possible successor states of s is denoted as succ(s). Applying a sequence of operators $o_1, o_2, \dots, o_n$ from state s will generate state $s[o_1][o_2]\cdots[o_n]$. A *s-plan* for Π is a finite sequence of operators such as $s[o_1][o_2]\cdots[o_n]$ is a goal state. A state s is *solvable* if a s-plan exists. The task Π is *solvable* if an s_0-plan exists. In this thesis, we will call "s_0-plan" just "plan".

In this work, we call any s_0-plan a *path* from s_0 to any goal state. A path is indicated as $s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$, where $s_i \rightarrow s_j$ indicates that s_j is a direct successor of s_i and s_n is a goal state.

Given a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*, \text{cost}\rangle$, we say that it induces a *state space* $\mathcal{S} = \langle s_0, S_*, \mathcal{O}, \text{cost}\rangle$, a weighted and labeled transition system, represented as a graph, where each node in this graph corresponds to a reachable state from s_0 and each arc from state s to s' corresponds to the operator o generating s from s' and it is weighted with cost(o).

Definition 2.3 (State Space). *Let $\mathcal{S} = \langle s_0, S_*, \mathcal{O}, cost\rangle$ be a* state space *induced by $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*, cost\rangle$, where s_0 is the initial state, S_* is a set of goal states (set of states that contain the partial variable assignment s_*) and $\mathcal{O}$ is a set of operators. For a given state s and for each operator o in the subset of applicable operators to s in $\mathcal{O}$, we have the transition $s \rightarrow s'$ with label* o *and cost cost$(o) \in \mathbb{R}_0^+$, where $s' = s[o]$.*

Any sequence of labels from transitions in $\mathcal{S}$ from s_0 to any state $s \in S_*$ correspond to a valid plan in Π. The shortest sequence corresponds to the *optimal plan* in Π. We denote the *cost of the optimal plan* as C^*.

2.2 Heuristic Functions

In this section, we formally introduce *heuristic functions*, or just called *heuristics*. In the following section, we will introduce search methods and how they use heuristics as defined here.

In the context of planning, heuristics can be seen as a function mapping states from a state space $\mathcal{S}$ to a non-negative number. Usually, a heuristic computed for a state

s estimates the cost from s to a goal state $s' \in S_*$.

Definition 2.4 (Heuristic Function). *Let $\mathcal{S} = \langle s_0, S_*, \mathcal{O}, cost\rangle$ be a state space induced by $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*, cost\rangle$, the* heuristic function *(or just* heuristic*) is a function $h : \mathcal{S} \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$.*

A special heuristic function is the *perfect heuristic*, denoted as h^*. This function maps each state $s \in \mathcal{S}$ to the cost of an optimal s-plan in Π (i.e., it estimates perfectly the minimum cost for each state to reach a goal state in a given state space).

Definition 2.5 (Perfect Heuristic). *Let $\mathcal{S} = \langle s_0, S_*, \mathcal{O}, cost\rangle$ be a state space induced by $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*, cost\rangle$, the function $h^* : \mathcal{S} \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$, called the* perfect heuristic*, maps each state $s \in \mathcal{S}$ to the cost of an optimal s-plan in Π or to ∞ if no such plan exists.*

It is also useful to introduce some properties of heuristics, such as *admissibility* and *consistency*:

Definition 2.6 (Admissible Heuristic). *Let h be a heuristic function over a state space $\mathcal{S} = \langle s_0, S_*, \mathcal{O}, cost\rangle$. The function h is said* admissible *if $h(s) \leq h^*(s)$ for every $s \in \mathcal{S}$.*

Definition 2.7 (Consistent Heuristic). *Let h be a heuristic function over the state space $\mathcal{S} = \langle s_0, S_*, \mathcal{O}, cost\rangle$ induced by $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*, cost\rangle$. The function h is said* consistent *if and only if $h(s) \leq h(s') + cost(o)$ for every $s \in \mathcal{S}$ and each $s' \in succ(s)$ where $s' = s[o]$ for an applicable operator $o \in \mathcal{O}$; and $h(s_*) = 0$ for each $s_* \in S_*$.*

For a state space $\mathcal{S}$ and a heuristic function h, we can define a *state space topology $\mathcal{T}$*. In words, a state space topology is a state space where each state is evaluated accordingly to a heuristic function h.

Definition 2.8 (State Space Topology). *Let h be a heuristic function over the state space $\mathcal{S} = \langle s_0, S_*, \mathcal{O}, cost\rangle$ induced by $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*, cost\rangle$, a* state space topology *$\mathcal{T} = \langle \mathcal{S}, h\rangle$ is a state space where each state $s \in \mathcal{S}$ is evaluated and labeled accordingly to the function h.*

In planning literature, heuristics were usually divided in four types: delete relaxation, critical paths, landmarks and abstractions (HELMERT; DOMSHLAK, 2009). These four types of heuristic will be explained in the following subsections. More recently, flow (BRIEL et al., 2007; BONET, 2013) and potential heuristics (POMMERENING et al., 2015) were introduced as “new types”; however, they are not fundamental to the results of this thesis, thus they won’t be detailed.

2.2.1 Delete Relaxation

Heuristics based on delete relaxation "accumulate" the effects of its operators instead of changing the facts in a state. Using the notation from the literature, we denote the delete relaxation heuristic as h^+. We illustrate this concept with a small example.

Example 2.1. Suppose we have a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*, \text{cost} \rangle$ where $\mathcal{V} = \{X, Y, Z\}$, with $\text{dom}(X) = \text{dom}(Y) = \text{dom}(Z) = \{0, 1\}$; the initial state s_0 is

$$s_0 := \{X = 1, Y = 0, Z = 0\}$$

The goal partial variable assignment is:

$$s_* := \{Y = 1, Z = 1\}$$

Our operator set $\mathcal{O}$ has three operators o_1, o_2 and o_3:

$$\begin{aligned} o_1 &:= \langle \{X = 1\}, \{Y = 1, Z = 1\} \rangle \\ o_2 &:= \langle \{X = 1\}, \{X = 0, Y = 1\} \rangle \\ o_3 &:= \langle \{X = 1\}, \{X = 0, Z = 1\} \rangle \end{aligned}$$

And our cost function maps the operators as follows:

$$\begin{aligned} \text{cost}(o_1) &= 1 \\ \text{cost}(o_2) = \text{cost}(o_3) &= 0 \end{aligned}$$

Computing the perfect heuristic for our initial state we would have $h^*(s_0) = 1$, because applying the o_1 would lead us directly to a goal state. However, we would also obtain $h^+ = 0$, because the plan over delete relaxation would allow us to apply o_2 ans o_3 to reach a goal state – even though it is not even feasible in our real original task. In more detail, the following would happen:

$$\begin{aligned} s_0 &:= \{X = 1, Y = 0, Z = 0\} \\ s_0[o_2] &:= \{X = 1, X = 0, Y = 1, Y = 0, Z = 0\} \\ s_0[o_2][o_3] &:= \{X = 1, X = 0, Y = 1, Y = 0, Z = 1, Z = 0\} \end{aligned}$$

and $s_0[o_2][o_3]$ is a goal state. $\square$

Computing h^+ is NP-complete (BYLANDER, 1994; BONET; GEFFNER, 2001), however, we can compute approximations of h^+ in polynomial time. Even though delete relaxation heuristics don't seem very promising in our example, approximations of h^+ won the International Planning Competition (IPC) in previous years (BONET; GEFFNER, 2001; HOFFMANN; NEBEL, 2001). It is worth to mention that h^+ is admissible (BONET; GEFFNER, 2001).

The most simple approximation of h^+ is h^{max} (BONET; GEFFNER, 2001). In general lines, the function h^{max} computes the maximum cost to achieve individually each fact in s_* in the delete relaxation version of the task. The computation can be performed in polynomial time and the heuristic value is admissible. However, as one may expect, h^{max} is not very informative in practice. One can consider h^{max} as an *optimistic* view of h^+; another possibility is the *pessimistic* view, the h^{add} heuristic (BONET; GEFFNER, 2001). It also computes the individual cost for each fact in s_* in the delete relaxation planning task, but instead of taking the maximum cost, it sums them all. By definition, h^{add} is not admissible and not bounded by h^+. h^{add} will not be explained further because it won't appear in this work again.

Another important heuristic is h^{FF} (HOFFMANN; NEBEL, 2001). In 2001, Hoffmann and Nebel (2001) introduced this idea and won the IPC using a planner based on it. Since then, the h^{FF} heuristic is one of the most relevant approximations of delete relaxation in the planning literature. This heuristic uses the observation that relaxed solvability of a delete relaxation is in P (BYLANDER, 1994) to find a feasible plan quicker. Here, we remark two results are important for this work: *(i)* h^{FF} is non-admissible; and *(ii)* h^{FF} is at least as informed as h^{max} in any given $\mathcal{S}$ (HOFFMANN; NEBEL, 2001).

Given a planning task Π accordingly to Definition 2.2, we will denote as Π^+ the *delete relaxation* of Π.

2.2.2 Critical Paths

We call *critical path heuristics* the family of heuristics h^m defined by Haslmum and Geffner (2000). The heuristic h^m computes the lower bound cost to achieve a set of goal facts of cardinality m, where $m \in \mathbb{N}_1$ is a parameter. For a given state, it can be computed in polynomial time for a fixed m, but exponential time in m. One important point about critical path heuristics is that its basic form ($m = 1$) is equivalent to h^{max}.

2.2.3 Abstractions

For a given state space $\mathcal{S}$, an *abstract heuristic* h^α maps each state $s \in \mathcal{S}$ an *abstract state* $\alpha(s)$. Thus, the heuristic value $h^\alpha(s)$ is the estimated cost from s to the closest abstract goal state in the transition system induced by α over $\mathcal{S}$.

One of the main abstraction mappings in literature are *pattern databases* (CULBERSON; SCHAEFFER, 1998; EDELKAMP, 2001). Roughly speaking, in pattern databases abstractions (PDB) we relax the planning task Π to consider only a subset of variables from Π and ignore the other ones – we simply "don't care" about their values. Hence, we compute the heuristic value for each abstract state only considering these variables – e.g. if an operator has a precondition not related to a variable in this subset, we discard it.

A PDB heuristic stores the shortest distance to the closest abstract goal state for all abstract states in a look-up table. For computing the heuristic value of a state, we query the look-up table for the abstraction which corresponds to the original state to get a *lower-bound* on the cost to the closest goal state. Thus, it can be used as an admissible heuristic. In most of the cases, it is interesting to compute several different PDBs for a planning task and get the maximum among them. Given some special conditions, we can even sum up different PDB heuristics for a same state (In a nutshell, both abstractions must be *disjoint* – i.e., the variables in each subset must be different and the operators used to estimate each heuristic value must only depend on variables from one abstraction).

In this work, we will use mostly the iPDB heuristic (also written h^{iPDB} (HASLUM et al., 2007; SIEVERS; ORTLIEB; HELMERT, 2012). The details of iPDB are omitted and further results will be shown in specific sections later. Other possible abstractions are *merge-and-shrink* abstractions (HELMERT et al., 2007) and *structural patterns* (KATZ; DOMSHLAK, 2008), the former will be used in some experiments but its details are not crucial.

2.2.4 Landmarks

For a state s, we say that an operator $o \in \mathcal{O}$ is an *action landmark* if it is part of any s-plan. *Disjunctive action landmarks* are sets of actions where at least one is part of every s-plan. We also have *fact landmarks* for s: facts that must be true in any s-plan. In this thesis, we refer as just *landmarks* the notion of disjunctive action landmarks for a given state s.

Deciding if a set of operators $\mathfrak{L} \subseteq \mathcal{O}$ is a landmark in a planning task Π is PSPACE-complete (PORTEOUS; SEBASTIA; HOFFMANN, 2014). Hence, most of the heuristics in literature use relaxations of the planning task to compute $\mathfrak{L}$. In practice, we usually use the delete relaxation Π^+ to compute a set $\mathfrak{L}^+$. Therefore, these heuristics are also bounded by h^+ (HELMERT; DOMSHLAK, 2009).

Several heuristics using landmarks were proposed (e.g. RICHTER; HELMERT; WESTPHAL (2008) and KARPAS; DOMSHLAK (2009)). Helmert and Domshlak (2009) define the *landmark-cut* admissible heuristic function $h^{\text{LM-cut}}$, or simply LM-cut. The details of this heuristic are outside of the scope of this thesis.

Now, we present heuristic search techniques and how we can use the heuristics presented previously to help us to find a solution to our planning tasks quicker.

2.3 Heuristic Search

Table 2.1: Cases where A* is 0- or 1-optimal.

Characteristics of h in Π	$\mathrm{A} \land \mathrm{P}$	$\mathrm{A} \land \neg\mathrm{P}$	$\mathrm{C} \land \mathrm{P}$	$\mathrm{C} \land \neg\mathrm{P}$
Characteristics of A*	1-optimal	0-optimal	1-optimal	0-optimal

In a summary, most of the deterministic search algorithms work as follows: given a graph $G = (V, A)$, we select our initial node $n \in V$ and generate all possible successors of n – we say that we *expand* n by applying all possibles transitions to it – and then we place them in a list called OPEN. After the expansion, we place n in a list called CLOSED. Next, we must select the next node from OPEN to expand and apply this procedure successively until we find a goal node, the node that we are searching for (it can be when we generate or when we select it from OPEN, depending on the specific algorithm). The main difference between search algorithms is the way we select the next node to expand. For instance, the two most basic search algorithms are breadth-first search and depth-first search; while the former uses a first-in-first-out (FIFO) fashion, the latter uses a last-in-first-out (LIFO) manner. These different selection methods can cause a difference in performance and in the optimality of the search.

In the context of planning, each node of the search is equivalent to a state. Thus, we say that we perform a *state-space search* over a state space $\mathcal{S}$. Expanding a state s is the same as applying all applicable operators in this state to generate $\text{succ}(s)$. We decide to use the term "state" instead of "node" through the rest of this work; however, we must mention that they could be interchanged without loss of meaning.

Planning as heuristic search is mostly based on best-first search algorithms. This class of algorithms uses the heuristic information to evaluate a function f for each state s and then place it in the OPEN list. When selecting which state to expand, the algorithm selects the state in OPEN which minimizes the function f. Thus, we try to perform our state-space search leading forward to a goal state, minimizing the number of states expanded and making the search faster.

A* is a best-first search algorithm that uses $f(s) = g(s) + h(s)$ for all states s. The function $g(s)$ denotes the current cost until s from our initial state s_0 and $h(s)$ is the heuristic function over s. The function $f^*(s)$ denotes the f-value when using the perfect heuristic h^* for its computation – i.e., $f^* = g + h^*$. Also, we denote as $d(s)$ the function that computes the number of operators needed from a state s until the closest goal state in an optimal plan.

Two properties of A* make it "special": (i) if the heuristic function h is admissible – i.e., it never overestimates the real cost to reach a state –, A* will find a plan with optimal cost $\mathcal{C}^*$; (ii) A* is said *optimal* – i.e., expand the fewest number of nodes – among best-first search algorithms that return optimal cost solutions, if the function h used is admissible and consistent (DECHTER; PEARL, 1985).

When studying the properties of A*, Dechter and Pearl (1985) define a state space topology $\mathcal{T} = \langle \mathcal{S}, h \rangle$ to be *non-pathological* if there exists at least one cost-optimal, not fully informed s-path – an s-path is not fully informed if $h(s) < h^*(s)$ for all $s \notin S_*$ on it. In the same way, a *pathological* state space topology $\mathcal{T}$ is fully informed on at least one state $s \notin S_*$ on each cost-optimal s-path.

Definition 2.9 (Pathological State Space Topology)**.** *A state space topology* $\mathcal{T} = \langle \mathcal{S}, h \rangle$

is pathological *if, for each cost-optimal s-path, at least one state $s' \notin S_*$ on the s-path has* $h(s') = h^*(s')$.

We analyze further the results from Dechter and Pearl (1985) to use it as a motivation for part of this thesis. When compared to best-first search algorithms, A* is said 0-optimal if it expands a subset of the states expanded by any other best-first search independently of the tie-breaker, and it is said 1-optimal if there is a tie-breaking strategy in which A* expands a subset of the states expanded by any other best-first search algorithm. Table 2.1 show in which circumstances A* is 0- and 1-optimal from Dechter and Pearl (1985). The analysis is made given a specific task Π and a heuristic h. “A” indicates a task where h is admissible; “P” indicates a task where h is pathological and its negative, “$\neg$P”, when it is non-pathological; “C” indicates that h is consistent in Π. In this second part of this thesis, we will study tie-breaking techniques and hence we will be interested in the case where A* is 1-optimal.

Equipped with the definitions from this chapter, we move forward to analyze some problems in classical planning. We suggest the reader to consult the list of symbols during the following chapters, since the notation can be quite complicated to readers who are not used to automated planning or heuristic search.

3 THE AIRPORT GROUND TRAFFIC PROBLEM

One of the planning problems that arises in airports is the ground traffic control problem, which is to define the paths out-bound airplanes have to take from their parking position to the runway for take-off, or the paths in-bound airplanes have to take after landing until getting to a parking position. The main objective is to find a plan that achieves this goal most economically, i.e., with the least movement of the airplanes, respecting restrictions on the movement coming from concerns about security, for example the minimum distance between airplanes.

A realistic version of the airport traffic control problem was introduced in the IPC in 2004 (HOFFMANN et al., 2006). In this version, the airport is modeled as a directed graph, with designated parking and take-off positions. All airplanes have an orientation and other airplanes must keep a minimum distance from the rear of an airplane with a running engine.

There are several versions of the airport ground traffic control problem. In the simplest version, all routes for in- or out-bound airplanes are fixed. This is often done to simplify the decision making of the ground traffic controller. However, even this version is NP-complete (HATZACK; NEBEL, 2001). If the routes are not prefixed, deciding if a bounded plan or any feasible plan exists is PSPACE-complete, even when the goal positions of the airplanes are known (HELMERT, 2006b). This is one of the reasons that makes the airport domain one of the most difficult domains in the IPC. Four versions have been introduced in IPC 2004 and they differ in the modeling of time. The simplest version is non-temporal. There also exist temporal versions with and without time windows. In practice the goal is to minimize the *makespan* of an instance, i.e., the total time or number of operators until all airplanes reach their goal positions.

In this thesis, we are interested in solving the non-temporal, sequential version of the problem optimally, i.e., we search for plans which minimize the total number of operators. In the instances of IPC 2004, the goal positions of the airplanes (a runway for out-bound airplanes, and a parking position for in-bound airplanes) are fixed. However, we are also interested in a variant of the problem that permits free goal positions, such that in-bound airplanes can choose any of the available parking positions and out-bound airplanes can take off at several available runways. This variant is more realistic since, for example, the parking position of an airplane may change after touchdown. Besides being a more realistic variant, this change makes the problem also more difficult to solve, since for a_i in-bound airplanes, p parking positions, a_o out-bound airplanes and r runways the number of goal states increases from 1 to $\binom{p}{a_i}r^{a_o}$.

The techniques used by solvers for airport ground traffic problems usually are domain-independent. These techniques do not take into consideration particularities of the problem, and try to solve it only by general strategies on the planning task. The main goal of this chapter is to compare how well simple domain-dependent heuristics perform when compared to state-of-the-art domain-independent heuristics in both variants of the problem mentioned above.

Another motivation to study the airport domain is to analyze techniques used in other transport problems and see how well they perform using the automated planning formalism. We are specially interested to test domain-dependent heuristics similar to the ones used in Sokoban, (e.g. PEREIRA; RITT; BURIOL (2014)).

We close this introductory section mentioning that the following results and observations were already published in a past conference paper (CORRÊA; PEREIRA; RITT, 2016). Here, we extend our analysis to present more detailed observations and results.

3.1 Preliminaries

3.1.1 Airport Ground Traffic Control

Our definition of airport ground traffic control tasks is not in “pure” SAS$^+$, because this “pure” version can be messy and misleading for some readers. Instead, we use an easier – but correct – formalism to explain this domain. Our explanation is based on Helmert (2006b), the main differences are made to keep the notation consistent.

An airport task is defined by a set of segments ψ. Each segment $\psi \in \Psi$ can be either a taxiway, a parking position, or a runway for landing and take-off, and has domain dom(ψ) $= \{taxi, park, runway\}$. Thus, an airport can be modeled by a directed graph $G = (V, A)$, with vertex set $V = \Psi \times D$, where $D = \{north, south\}$ represents the possible directions of an airplane, and arc set A, which defines the possible successor segments and directions for a given current segment and direction[1].

A task of the ground traffic control problem is defined by a set of airplanes P, with positions $p : P \rightarrow V$ and goal segments $g : P \rightarrow \Psi$. The type of a goal segment $g(a)$

[1]Note that “*north*” and “*south*” are arbitrary names, introduced by Hoffmann et al. (2006), and have no relation to any actual orientation of the segment.

for an airplane $a \in P$ must be either *park* for in-bound airplanes or *runway* for out-bound airplanes. An airplane can be either *moving*, *parked* or *airborne*, defined by its current mode $m(a) \in M = \{\textit{parked}, \textit{moving}, \textit{airborne}\}$ (i.e., similar to $\mathrm{dom}(a)$). A position of an airplane $a \in P$ is defined by its current segment and its direction. The direction defines to which end of the segment the airplane is oriented, and restricts the possible successor segments

Mode *parked* is only permitted if the current segment of the airplane has type *park*. A *parked* airplane can start up its engines to enter mode *moving*. Only in this mode an airplane can change its current segment and direction (p, d) to another segment and direction (p', d') if $((p, d), (p', d')) \in A$. If an airplane is in mode *moving* at a segment of type *park* it can similarly shut down its engines to enter mode *parked*. Mode *airborne* is only valid for segments of type *runway*. An airplane in mode *moving* at a runway can take off to switch to mode *airborne*. There is no operator for landing. All operators in the airport domain have unit cost.

A *moving* airplane at position $v \in V$ will block a set of segments $B(v) \subseteq V$ due to the engine exhaust of the airplane. The blocked segments depend on the orientation of the airplane and its size (larger airplanes may block more segments). For current positions p and modes m, segments $B(p, m) = \cup_{a \in P^g} B(p(a))$ are blocked, where $P^g(m) = \{a \in P \mid m(a) \neq \textit{airborne}\}$ is the set of airplanes on the ground. Let $\Psi^g(p, m) = \{\psi \mid (\psi, d) = p(a) \text{ for some } a \in P^g(m)\}$ be the segments occupied by airplanes on the ground. The current positions p are valid, if on each segment there is a most one airplane of type *moving* or *parked* (i.e., $|\Psi^g(p, m)| = |P^g(m)|$), and no airplane on the ground is at a blocked segment (i.e., $\Psi^g(p, m) \cap B(p, m) = \emptyset$).

The standard airport domain with fixed goal positions can be modeled as a weighted state space $\mathcal{S} = \langle s_0, S_*, \mathcal{O}, \mathrm{cost}\rangle$, as defined before. Our initial state s_0 specifies in which segments each airplace is located, which is the type of each airplane, which segments are blocked, where are the parking and takeoff segments, etc. The set of operators $\mathcal{O}$ contains operators to move airplanes (forward and backwards), turn planes on and off, as well as operators to park and takeoff airplanes. All the operators have unit cost in both versions of this domain. The only difference between the two presented versions is the goal S_*: the original version with fixed parking positions has a specific fact for each plane in the format $\langle$`is-parked-?a-?s`$, b\rangle$, where b is a boolean variable, and the variable `is-parked-?a-?s` indicates if airplane `?a` is parked at segment `?s` (the format for airborne planes is similar); the new version with free parking position has goal facts in the format $\langle$`is-parked-?a`$, b\rangle$ simply indicating if airplane `?a` is parked or not (the airborne planes keep the goal from the original version)[2].

3.1.2 Standard Set of Instancess

The standard set of instances of the airport domain represents real-world conditions of an airport ground traffic control problem. There are four versions of the domain: *non-temporal*, *temporal*, *temporal-timewindows*, and *temporal-timewindows-compiled*. The last three versions include time constraints, as for example the time to move across a segment and time windows where a segment will be blocked. We address the first version which is the most widely researched in the literature. In this IPC version, each task is written in Planning Domain Definition Language (PDDL). PDDL is similar to SAS$^+$, however not identical. The facts in SAS$^+$ are called *predicates* and they can have multiple parameters. For example, the fact to check if a goal is parked in SAS$^+$ explained in the previous subsection was $\langle$`is-parked-?a-?s`$, b\rangle$, where `is-parked-?a-?s` is a finite-domain variable indicating if airplane `?a` is parked at segment `?s`; in PDDL, we could write it just as `(is-parked ?a ?s)`, a predicate with two parameters – an airplane and a segment – and this predicate must be true in any goal state.

In the standard set, for each instance there is a unique explicitly defined goal state. The goal state is defined by a set of two predicates `(is-parked ?a ?s)` and `(airborne ?a ?s)`, where `?a` is an airplane and `?s` is a segment. Each predicate defines the segment that the airplane must reach to achieve the goal condition. There are two actions, `park` and `takeoff`, that make the predicates true. Thus, the transformation from fixed to free goal positions consists of removing from the `is-parked` and `airborne` predicates the segment definition. The predicate `(is-parked ?a ?s)` has been changed to `(is-parked ?a)` and a similar change has been applied to the predicate `airborne`. We did not add any other action to the domains and just adapted it to make it consistent. The resulting definition produces an implicit defined goal state, where usually multiple states will satisfy the goal condition.

The standard set from IPC-4 has 50 instances. Table 3.1 shows their characteristics. It reports, for each instance, its number ("#"), the airport type ("T."), the total number of segments ("#S"), the number of in-bound airplanes and parking positions

[2]We use the notation `?a` instead of a here to use the same fashion in which PDDL (explained in the next subsection) uses to define it.

Table 3.1: Characteristics of the 50 instances from the airport domain.

			Park.		Airb.					Park.		Airb.	
#	T.	#S	#P	#S	#P	#S	#	T.	#S	#P	#S	#P	#S
1	1	17	1	1	0	2	26	4	302	3	30	3	2
2	1	17	0	1	1	2	27	4	302	1	30	5	2
3	1	17	1	1	1	2	28	4	302	2	30	5	2
4	2	40	1	2	0	2	29	4	302	3	30	5	2
5	2	40	0	2	1	2	30	4	302	1	30	7	2
6	2	40	1	2	1	2	31	4	302	2	30	7	2
7	2	40	1	2	1	2	32	4	302	3	30	7	2
8	2	40	1	2	2	2	33	4	302	2	30	8	2
9	2	40	2	2	2	2	34	4	302	3	30	8	2
10	3	44	1	4	0	2	35	4	302	4	30	8	2
11	3	44	0	4	1	2	36	5	457	1	30	1	4
12	3	44	1	4	1	2	37	5	457	0	30	3	4
13	3	44	1	4	1	2	38	5	457	1	30	2	4
14	3	44	1	4	2	2	39	5	457	0	30	4	4
15	3	44	1	4	2	2	40	5	457	1	30	3	4
16	3	44	1	4	3	2	41	5	457	2	30	2	4
17	3	44	2	4	3	2	42	5	457	1	30	4	4
18	3	44	2	4	4	2	43	5	457	2	30	3	4
19	3	44	3	4	3	2	44	5	457	3	30	2	4
20	3	44	3	4	4	2	45	5	457	4	30	2	4
21	4	302	1	30	1	2	46	5	457	2	30	4	4
22	4	302	2	30	1	2	47	5	457	4	30	4	4
23	4	302	3	30	1	2	48	5	457	2	30	7	4
24	4	302	1	30	3	2	49	5	457	4	30	6	4
25	4	302	2	30	3	2	50	5	457	4	30	11	4

(“Park./#P”,“Park./#S”), and the number of out-bound airplanes and take-off positions (“Airb./#P”,“Airb./#S”). Airports of the same type have the same structure, and thus the same number of segments. There are five types of airports with an increasing number of segments. The two largest airport types are realistic encodings of the Munich airport. The smallest airport has 17 segments, the largest one has 457. The maximum number of planes is 15. In 31 of the 50 instances there are more airplanes which are out-bound than in-bound, and that the number of goal segments for out-bound airplanes is usually small, and never more than four. When considering the implicit defined goal state, the effect of multiple goal states in the first three types of airports is limited. The situation changes in the more realistic airports of type 4 and 5. In these cases we have still a small number of 1 to 4 of in-bound airplanes, but 30 possible parking positions, and thus up to $\binom{30}{4}$ possible goal states for the in-bound airplanes.

3.1.3 Related Work

General traffic control problems have been studied by Hatzack and Nebel (2001). They proposed the only domain-dependent heuristic solution for the problem we are aware of. They also have showed that the problem with fixed routes is equivalent to a job shop scheduling problem which has to satisfy blocking conditions without swaps. Blocking constraints in job shop scheduling are less common. When reducing traffic control problems to job shop problems, these constraints model the fact that a resource occupied by a vehicle can be used by another vehicle only if the first moves on to an adjacent resource, and if two vehicles which meet head-on at two adjacent resources, they cannot simply change places, since this would lead to a collision. The reduction to job shop scheduling also shows that the problem with fixed paths still remains NP-complete. Their algorithm schedules the vehicles in order of non-increasing release time, and starts each operation greedily at the first possible time after its release or the completion of its predecessor. The airport ground control traffic problem has been also studied in the PhD thesis of Hatzack (2002).

Trüg, Hoffmann and Nebel (2004) concluded that the best planners in that time were not yet able to solve real-world airport instances. They also observe that the core of the problem is to resolve the conflicts that arise when several airplanes have to access the same segments and that the automated planners are not aware of this fact.

The problem considered in this work can be seen as a particular instance of multi-agent pathfinding problems with additional restrictions (e.g. mutual blocking of airplanes by their jet stream, see below). Thus it seems possible that existing algorithms, e.g. *Push and Rotate* (WILDE; MORS; WITTEVEEN, 2014), may be adapted to solve it. There are however, fewer approaches for solving such problems optimally (STANDLEY, 2010; STANDLEY; KORF, 2011; SHARON et al., 2011; SHARON et al., 2012).

Our interest in the airport also stems from its similarities with other transportation problems with a large number of goal states, such Sokoban. Apart from airport-specific blocking rules, both problems share the main difficulties which come from the interaction of movable objects, and, for the variant of the airport ground traffic control problem with free goal positions, from the $k!$ goal states. More generally, a larger class of block-moving games shares these characteristics. In a previous work was identified the weakness of applying pattern databases to problems with lots of goal states, and a decomposition of the task to overcome it was proposed (PEREIRA; RITT; BURIOL, 2013; PEREIRA;

RITT; BURIOL, 2014; PEREIRA; RITT; BURIOL, 2015). Here, we are interested to see if this weakness applies to the airport ground traffic control problem, and if it may be overcome with similar techniques.

3.2 A Domain-Dependent Heuristic for the Airport Domain

A domain-independent heuristic does not have prior knowledge about the structure of the problem. Consider, for example, a domain-independent pattern database heuristic. In the airport domain such heuristic would have to compute for each airplane the distance from every segment to every goal position by a reverse search. However, we know that the distances for all airplanes are the same, independent of other characteristics such as the airplane's size. Thus, we can compute the distances only once and reuse them for all airplanes. This is an example where we can use knowledge about the problem structure to improve the efficiency of the heuristic. Our first proposed heuristic comes from reusing these distances in a very simple way.

In the version of the airport domain with free goal positions an in-bound airplane can park at any parking segment. This is similar to Sokoban, where a stone must be pushed to one of many goal squares. However, in the goal state each at goal square must be exactly one stone. The standard approach to compute the heuristic function in Sokoban is to compute a minimum cost perfect matching between stones and goal squares, covering all goal squares with stones. Inspired by this approach, we propose a matching based heuristic for the Airport Domain.

3.2.1 Pre-Processing

We extract a directed graph for each task whose vertices represent the segments of the airport. Two segments (u, v) are connected if there is an operator that moves an airplane from a segment u to segment v. We then compute the shortest path between all pairs of segments of this graph and store the distances in a look-up table. This is done in a pre-processing phase, and all airplanes use the same distances during the search to compute the heuristic values. These distances are the shortest path between every pair of segments without taking into account path conflicts, blocked segments or any other interactions between the airplanes.

3.2.2 Closest Goal Heuristic

Our first approach is the domain-dependent closest goal heuristic. The closest goal heuristic is the sum of the distances required for all airplanes to reach their goal positions using the pre-computed distances. For the version with fixed goal positions, we look up the distance for each airplane to reach its specific goal position. In the version with free goal positions, we have to find the closest goal position for each airplane. For both versions, the cost for computing the heuristic is linear in the number of airplanes, since we can pre-compute the distances in constant time and then calculate the closest goal position for each airplane in each state. Additionally, each out-bound airplane must start up its engine and finally take off. Thus we increase the heuristic by one or two for each out-bound airplaine according to its current mode. Similarly, in-bound airplanes must shut down their engine to park, so we increase the heuristic by one for each in-bound airplane whose engine is still running.

In the version with fixed goal positions, with a_i in-bound airplanes the closest goal heuristic returns the cost to cover a_i parking positions. This is not the case in the version with free goal positions. In the latter version, all a_i airplanes can be mapped to the same parking position, which is not a valid solution. In this case, we could lose up to the sum of the $a_i - 1$ farthest parking positions in the heuristic value.

3.2.3 Matching Heuristic

Another natural heuristic to solve the airport domain problem is based on matchings. To compute the minimum distance for all airplanes to reach a different goal position, we construct a bipartite graph. In this graph one part of the vertices represents the airplanes and the other part of the vertices represents all possible goal positions. Between each airplane and goal position, we add an edge with a weight corresponding to the length of the shortest path the airplane can take to reach that goal position. Since the number of goal positions exceeds the number of airplanes, we add a suitable number of dummy airplanes to make the cardinality of the two parts equal. These airplanes have distances of 0 to each goal position. Then a minimum weight perfect matching of airplanes and goal positions is a lower bound on the number of moves needed to bring each airplane to a different goal position.

We compute the minimum perfect matching using the Blossom V algorithm (KOL-

MOGOROV, 2009) and also using the pre-processed distance information extracted from the instances. The matching can be seen as an extension of the closest goal approach. The difference is that while the closest goal heuristic tries to minimize the individual distance for each airplane to its goal position without taking into account multiple airplanes occupying to the same goal position, the matching heuristic tries to minimize the sum of the distances ensuring that two airplanes cannot be moved to a same goal position. As for the closest goal heuristic, we can increase the heuristic value by the number of outstanding engine shutdown operations.

The algorithm considers only airplanes that must be parked. Since the segments of out-bound airplanes will become available for other airplanes after take-off, every out-bound airplane could head for the same runway segment. We can notice that in the fixed goal positions variant of the problem the minimum perfect matching is trivial, since each vertex of the bipartite graph would have only one edge and would be equivalent to the closest goal approach.

3.3 Experimental Results

In this section we report computational experiments comparing domain-independent and domain-dependent heuristics on the airport domain with fixed and free goal positions. For the experiments we use the stable version 1.4 of the Fast Downward planner (HELMERT, 2006a). We have chosen Fast Downward for our implementation to be able to compare better to the domain-dependent heuristics, and have implemented our domain-dependent heuristics within the framework provided by the planner. All experiments have been run on a PC with an AMD FX-8150 processor running at 3.6 GHz and 32 GB of main memory. All domain-independent heuristics have been run with their default parameters. We have imposed a time limit of 30 minutes and a memory limit of 4 GB for each run.

3.3.1 Evaluation of domain-independent heuristics

In our first experiment we evaluate standard domain-independent heuristics on the airport domain. All heuristics have been run with their default parameters. Table 3.2

Table 3.2: Results for domain-independent heuristics.

	Fixed paths		Free paths	
Heur.	Cov. (50)	Nodes	Cov. (50)	Nodes
blind	22	1925645.7	23	2356855.9
$h^{\max}$	23	191425.6	23	417427.0
LM-cut	28	2508.2	26	19154.3
M&S	18	669808.3	18	910637.7
GAPDB	25	808624.5	27	1591448.8
iPDB	28	1594.0	28	68120.0

shows the results for blind heuristic[3], the heuristics $h^{\max}$ (BONET; GEFFNER, 2001), landmark-cut (HELMERT; DOMSHLAK, 2009) ($h^{\text{LM-cut}}$ or simply LM-cut), merge-and-shrink (HELMERT; HASLUM; HOFFMANN, 2007) (M&S), and two heuristics based on pattern databases: a pattern database which uses a genetic algorithm to create the patterns ("GAPDB") (EDELKAMP, 2007), and the iPDB (h^{iPDB} or simply iPDB) (HASLUM et al., 2007; SIEVERS; ORTLIEB; HELMERT, 2012). For each heuristic we report the coverage ("Cov.") – i.e., number of instances solved –, and the average number of explored nodes ("Nodes") over the optimally solved instances. Best results for each column are painted.

The standard domain-independent heuristics are able to solve between 18 and 28 instances. Heuristics $h^{\text{LM-cut}}$ and h^{iPDB} perform best for fixed as well as free goal positions. We can see that all heuristics need more nodes (and consequently more time) to solve the instances with free goal positions, as expected. However, this is not reflected in the number of solved instances. While $h^{\text{LM-cut}}$ solves two instances less, h^{iPDB} solves the same number of instances, and blind search and the GAPDB heuristic solve even more. This happens mostly in instances with a high number of out-bound airplanes, since the relaxation of the take-off positions simplifies the problem.

3.3.2 The closest goal heuristic on instances with fixed goal positions

In our second experiment we have evaluated the closest goal heuristic on the instances with fixed goal positions. We compared the results for the two best standard heuristics $h^{\text{LM-cut}}$ and h^{iPDB}, and the closest goal heuristic.

[3]The *blind heuristic* in the Fast-Downward planner is not equivalent to a *blind search* as used in literature. Usually, blind search means using a search algorithm without an heuristic – i.e., BFS, DFS, Dijkstra, etc –, however the blind heuristic h^{blind} returns the cost of the cheapest action if the current state is not a goal and 0, otherwise.

Table 3.3: Comparison of the closest goal heuristic to the two best domain-independent heuristics on the instances with fixed parking positions.

	LM-cut		iPDB		Closest	
Inst. (50)	Nodes	t	Nodes	t	Nodes	t
1†	9	0	9	0	9	0
2†	10	0	10	0	11	0
3†	20	0	24	0	34	0
4†	21	0	21	0	21	0
5†	22	0	22	0	23	0
6†	42	0	42	2	43	0
7†	65	0	82	2	112	0
8†	120	0	246	8	633	0
9†	767	1	2381	3	7456	0
10†	19	0	19	0	19	0
11†	22	0	22	0	23	0
12†	40	0	40	3	41	0
13†	50	0	61	2	81	0
14†	61	0	61	9	65	0
15†	155	0	474	13	1150	0
16†	402	1	2604	118	10571	0
17†	2484	14	34260	431	136373	3
18	15968	139	-	>	1025683	40
19†	2984	13	2263	166	61795	2
20	27433	279	-	>	359906	18
21†	102	0	102	18	103	0
22†	330	5	149	51	39479	1
23	351	14	169	721	1168583	51
24	164	7	163	1466	223	0
27	1508	222	-	>	2079	0
30	-	>	-	>	585630	119
36†	110	1	110	29	110	0
37	7954	242	148	309	36916	2
38†	9016	305	156	268	38855	2
39	-	>	616	1223	6151593	641
40	-	>	197	1482	1802028	188
41	-	>	180	875	937907	90
Tot.	28	8443	28	14399	**32**	**1165**

Table 3.3 reports the number of nodes ("Nodes") and the time ("t") for $h^{\text{LM-cut}}$, h^{iPDB} and the closest goal heuristic. We show only instances which were solved by at least one approach; instances marked with † were also solved by blind search. The last row of Table 3.3 shows the coverage and total search time for all instances. The closest goal heuristic solves 32 instances, four more than $h^{\text{LM-cut}}$ and h^{iPDB}. The solution time of closest goal is always less (with the exception of instance 23) and often by an order of magnitude. Comparing only instances which were solved by all three methods, the closest

Table 3.4: Values of the initial heuristic (h) and best f-value (f) for instances with fixed parking positions.

	LM-cut		iPDB		Closest	
Inst. (50)	h	f	h	f	h	f
31	358	358	357	357	358	358
32	390	390	387	387	390	390
33	393	393	388	388	393	393
34	427	427	422	422	427	427
35	433	433	425	425	433	433
39	208	208	210	210	208	210
40	190	190	191	191	190	191
41	178	178	179	179	178	179
42	257	257	259	259	257	257
43	223	223	224	224	223	223
44	227	227	229	229	227	227
45	249	249	251	251	249	249
46	290	290	292	292	290	290
48	423	423	399	399	423	423
49	451	451	447	447	451	451
50	685	685	670	670	685	685
Avg.	**301.9**	301.9	299.8	299.8	**301.9**	**302.1**

goal heuristic needs only $61s$ while $h^{\text{LM-cut}}$ needs $603s$ and h^{iPDB} $3619s$. However, $h^{\text{LM-cut}}$ and h^{iPDB} expand significantly fewer nodes than the closest goal heuristic.

We report in Table 3.4 the initial heuristic value ("h") and the best f-value ("f") – the lowest f-value on the open list when the time limit is reached or the task is solved – for those instances presenting different initial h or best f-values and that could not be solved by at least one method, . Best initial heuristic values or best f-values are highlighted. Instances for iPDB that were not solved and present f-value equals to initial heuristic value reached the time limit before the conclusion of the pattern database construction. We can notice that $h^{\text{LM-cut}}$ and the closest goal heuristic are usually better informed than h^{iPDB} in the beginning of the search. However, the closest goal initial heuristic value is weaker than h^{iPDB} when the number of airplanes is relatively small compared to the number of goal segments, but it does not necessarily impact the final f-value. Therefore, the initial h-value for closest goal and $h^{\text{LM-cut}}$ get much better as the number of goal variables increases (larger tasks). The only three instances where the final f-value for the closest goal heuristic is greater than the one for $h^{\text{LM-cut}}$ are the hardest instances which the closest goal heuristic can solve – i.e., tasks 39, 40 and 41 are the one which expand more nodes. We suspect that since our method is quicker than $h^{\text{LM-cut}}$, this speed up lead the search to explore the f-plateaus faster. All the instances that were not solved reached the time limit for the three methods.

3.3.3 The closest goal and the matching heuristic on instances with free goal positions

Table 3.5: Comparison of the closest goal and matching heuristics to the two best domain-independent heuristics on the instances with free parking positions.

	LM-cut		iPDB		Closest		Matching	
Inst. (50)	Nodes	t	Nodes	t	Nodes	t	Nodes	t
1†	9	0	9	0	9	0	9	0
2†	10	0	10	0	11	0	11	0
3†	20	0	24	0	34	0	34	0
4†	21	0	21	0	21	0	21	0
5†	22	0	22	0	23	0	23	0
6†	65	0	42	2	43	0	43	0
7†	65	0	43	2	45	0	45	0
8†	425	0	247	7	639	0	639	0
9†	793	2	2399	39	7559	0	6982	0
10†	19	0	19	0	19	0	19	0
11†	22	0	22	0	23	0	23	0
12†	40	0	40	2	41	0	41	0
13†	50	0	39	1	41	0	41	0
14†	61	0	40	2	65	0	65	0
15†	155	0	64	14	105	0	105	0
16†	402	1	94	123	307	0	307	0
17†	1766	11	321	436	4423	0	4337	0
18	8358	81	119075	1557	1102988	68	1101462	63
19†	419480	1194	763101	155	1722408	70	55108	3
20	-	>	-	>	-	>	569285	41
21†	96	1	96	10	97	0	97	0
22†	54311	596	54368	56	67148	6	146	0
23	-	>	966423	1513	1220965	170	150	0
24	158	7	157	585	217	0	217	0
25	-	>	-	>	-	>	137419	49
26	-	>	-	>	-	>	295878	211
27	1502	269	-	>	2073	1	2073	1
36†	79	1	79	22	79	0	79	0
37†	5435	185	154	224	22602	4	22602	12
38†	4648	1323	118	261	21075	3	21075	9
39	-	>	-	>	1644337	374	1644337	977
40	-	>	197	752	982983	277	982983	581
41	-	>	136	843	399723	99	399723	203
Tot.	26	15083	28	15616	30	6478	**33**	**2158**

In the third experiment we have evaluated the closest goal and the matching heuristic on the instances with free goal positions. Table 3.5 reports the number of nodes

Table 3.6: Values of the initial heuristic (h) and best f-value (f) for instances with free parking positions.

	LM-cut		iPDB		Closest		Matching	
Inst. (50)	h	f	h	f	h	f	h	f
20	113	113	113	113	113	113	115	115
23	144	144	144	144	144	148	148	148
25	206	206	206	206	206	206	208	208
26	196	196	196	196	196	196	200	200
28	288	288	288	288	288	288	290	290
29	284	284	284	284	284	284	288	288
31	352	352	351	351	352	352	354	354
32	366	366	363	363	366	366	370	370
33	387	387	384	384	387	387	389	389
34	403	403	399	399	403	403	407	407
35	383	383	378	378	383	383	391	391
39	142	142	144	144	142	144	142	144
40	159	159	160	160	159	160	159	160
41	134	134	135	135	134	135	134	135
42	185	185	187	187	185	185	185	185
43	179	179	180	180	179	179	179	179
44	174	174	176	176	174	175	176	176
45	190	190	192	192	190	190	194	194
46	205	205	207	207	205	205	205	205
47	264	264	251	251	264	264	268	268
48	325	325	266	266	325	325	325	325
49	354	354	276	276	354	354	358	358
50	533	533	400	400	533	533	537	537
Avg	260.4	260.4	248.9	248.9	260.4	260.7	**262.6**	**262.8**

("Nodes") and the time ("t") for $h^{\text{LM-cut}}$, h^{iPDB}, the closest goal and matching heuristics on the tasks which were solved by at least one approach. Instances marked with † were also solved by blind search. In Table 3.6 we show the initial heuristic value ("h") and the final f-value ("f") for tasks which were not solved by at least one method. Best initial heuristic values or best f-values are highlighted. Once again, instances for iPDB that were not solved and present f-value equals to initial heuristic value reached the time limit before the conclusion of the pattern database construction.

With free goal positions $h^{\text{LM-cut}}$ was able to solve 26, h^{iPDB} 28, closest goal 30 and matching 33 instances. As expected, we observe a degradation in performance of all methods, due to the larger number of goal states, and worse heuristic estimates. The performance degradation is limited, since the tasks have a small number of airplanes. There are only 25 tasks with more than two in-bound airplanes, and only 8 of them were solved with fixed goal positions, and the performance degradation is limited to the latter. This also explains why the matching heuristic is not substantially better than the closest goal heuristic. The difference will be only visible, for a larger numbers of in-bound airplanes. For example, in instance #39, which has no in-bound airplanes, both heuristics need the same number of nodes. In contrast, in instance #23, which has 3 in-bound airplanes, the matching heuristic is substantially better. However, it is remarkable that the matching heuristic dominates closest goal on the initial states and in instancess where the number of in-bound airplanes is strictly greater than the number of out-bound airplanes the matching heuristic also leads to a significantly higher lower bound than closest goal. The matching heuristic also expands fewer nodes than the closest goal heuristic when the number of in-bound planes is equal or greater than the number of out-bound airplanes.

In 47 of the 50 instances the matching heuristic presents the best final f-value over all methods. In 17 of the 25 instances presented in Table 3.6 the lower bound for matching is better than the best f-value for $h^{\text{LM-cut}}$ and closest goal. The same occurs for 16 instances when compared to h^{iPDB}. As the problem variant with fixed goal positions the initial heuristic value of h^{iPDB} is better than any other method when the number of airplanes is small compared to the number of goal segments. When the number of goal variables increases to 6 or more, iPDB becomes the weakest method. The closest goal heuristic is much less informative than the matching heuristic, but it performs better than h^{iPDB} and $h^{\text{LM-cut}}$ when comparing lower bounds and number of instances solved.

3.4 Conclusions and Future Work

We have proposed two domain-dependent heuristics for the airport domain and compared their performance to standard domain-independent heuristics in the Fast Downward planner. We found that even simple domain-dependent heuristic such as closest goal or matching can improve the results of state-of-the-art domain-independent heuristics. It is also remarkable to notice that in instances where the number of in-bound airplanes is larger than the number of out-bound airplanes the results of the closest goal and matching heuristics are significantly better than those of h^{iPDB} or $h^{\text{LM-cut}}$. In these instances the matching heuristic also presents better results than the closest goal heuristic as measured by the number of expanded nodes and initial and final lower bounds.

In future work, one could propose new instances increasing the number of in-bound planes this would present a more realistic version of the domain, since in real-world airports the number of in and out-bound airplanes should be approximately the same. In this more realistic version, given our current evidence, the heuristics h^{iPDB} and $h^{\text{LM-cut}}$ would likely have a greater performance degradation while the matching heuristic would maintain the same performance.

4 A PERFECT TIE-BREAKER FOR A*

A* is the most popular best-first heuristic search algorithm (HART; NILSSON; RAPHAEL, 1968). It expands states s in order of increasing f-values. Function $f(s)$ is the sum of the cost $g(s)$ of the current path from the initial state to state s, and the estimated cost $h(s)$ from s to a goal state. A heuristic h is *admissible* if it never overestimates the cost of a path from any states to some closest goal state. In this case A* returns an optimal solution path of minimum cost $\mathcal{C}^*$, if there is one. A heuristic h^* is *perfect* if it returns the cost of an optimal path for all states. During the search, it is possible to have several states with the same f-value. Then, A* uses a *tie-breaking strategy* τ to select one of them to be expanded next. A* with a deterministic tie-breaking strategy τ defines a unique *expansion sequence* of states.

A* must expand all states with $f(s) < \mathcal{C}^*$. Given a state space $\mathcal{S}$, a heuristic h is called *non-pathological* if there exists some cost-optimal path where $h(s) < h^*(s)$ for all non-goal states s on it. Dechter and Pearl (1985) have shown that if a heuristic h is non-pathological and admissible then the tie-breaker τ plays no role and the set of states with $f(s) < \mathcal{C}^*$ contains all states expanded by A*. Thus, in this case A* is optimal in the sense that it expands a subset of the states expanded by every admissible best-first search algorithm. If heuristic h is admissible and *pathological*, then A* will expand additionally some states s with $f(s) = \mathcal{C}^*$. This set of states is known as the *final plateau* or *final f-layer*. There is always a tie-breaking strategy τ that expands in addition to states with $f(s) < \mathcal{C}^*$ only states along a cost-optimal path with the least number of operators. For such a tie-breaker A* is again optimal in the number of expanded states.

Until recently, most of the search and planning literature considered breaking ties in favor of smaller h-values to be a good practice (ASAI; FUKUNAGA, 2016; ASAI; FUKUNAGA, 2017). Dechter and Pearl (1985) describe A* as being agnostic with regard to the tie-breaking strategy letting it "*break ties arbitrarily, but in favor of a goal state*" and assume that few states s will satisfy $f(s) = \mathcal{C}^*$. Helmert and Röger (2008) state that the number of states in the final plateau that A* expands depends on "*accidental features of the search algorithm*". However, recent work on tie-breaking strategies has shown that more instances can be solved and fewer states can be expanded by using tie-breaking strategies that do not favor small h-values directly (HEUSNER; KELLER; HELMERT, 2017; FAN; MÜLLER; HOLTE, 2017; ASAI; FUKUNAGA, 2017).

A special case where the final plateau accounts for a large part of the expanded states are *zero-cost instances* where some operators have cost zero (ASAI; FUKUNAGA, 2016). In many applications the goal is to minimize the use of some resource (e.g., fuel in logistic problems), and operators that do not use this resource can be modeled as having no cost. In these domains A* can follow very long zero-cost paths that can be avoided by a tie-breaking strategy.

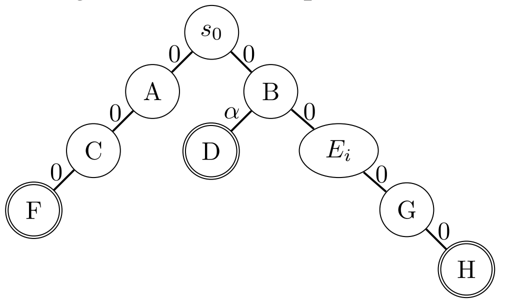
In this chapter, we analyze tie-breaking strategies for the perfect heuristic h^*. We first study previously proposed tie-breaking strategies and we prove that A* with h^* using these strategies is not optimal. We also propose a new one for which it is optimal. We experimentally analyze the performance of A* with h^* using several strategies on a restricted set of standard instances and instances with zero-cost operators. We also show that our strategies solve more instances than previous methods in the literature in a practical setting using A* with $h^{\text{LM-cut}}$. Our results show how to build an optimal A* for a perfect heuristic. Our analysis improves the understanding of how to develop effective tie-breaking strategies.

4.1 Tie-Breakers for A*

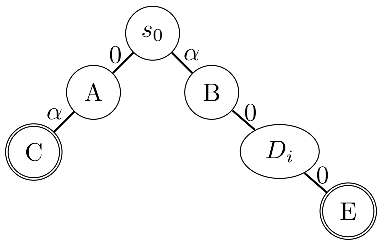
The A* algorithm receives a description of a state space topology $\mathcal{T}$ as input and outputs an s-path, if there is one, or "unsolvable" otherwise. Given $\mathcal{T} = \langle \mathcal{S}, h \rangle$, A* with a *tie-breaking strategy* τ expands a unique sequence of states $\langle s_0, s_1, \ldots, s_n \rangle$, called the *expansion sequence*. We assume that A* keeps a priority queue denoted as OPEN that sorts the states lexicographically in increasing order of $[f(s), \tau]$[1]. To expand a node means to generate all its successors. Note that in this way goal states are only removed from OPEN, but not expanded. If the expansion sequence of A* with a given tie-breaking strategy has the minimum number of states among all such sequences we say that it presents *optimal expansion*. A* is *optimal* if it has optimal expansion – i.e., it never expands a state that can be skipped by another admissible best-first search algorithm.

[1]We can also call $[f(s), \tau]$ as tie-breaking strategy. However, since we are analyzing only A* algorithm, the first evaluation function is always $f(s)$, hence we ommit $f(s)$ in some situations and mention τ as tie-breaking strategy

Figure 4.1: Search spaces for which tie-breaking by h^*, $\hat{h}^*$, and h^*_ϵ fails.



(a) Tie-breaking strategies $[g+h^*, h^*]$ and $[g+h^*, \hat{h}^*]$ fail.



(b) Tie-breaking strategy $[g + h^*, h^*_\epsilon]$ fails.

4.2 Understanding Tie-Breaking Strategies

In this section, we present a theoretical analysis of tie-breaking strategies for A*. Our analysis is based on the perfect heuristic h^*. In state spaces where we can compute h^*, A* will only expand states whose f-value equals the cost-optimal value $\mathcal{C}^*$. In this setting, A* will have optimal expansion if it only expands states on one cost-optimal s-path with the least number of operators.

4.2.1 Analyzing h^* as Tie-Breaking Strategy

The heuristic search literature usually considers breaking ties by h to be a good approach. Therefore one would expect that by having h^*, we could use its value as a tie-breaker, leading to an A* with optimal expansion. In this setting, $[g + h^*, h^*]$ denotes that A* uses $f = g + h^*$ and h^* as tie-breaker, any remaining ties are solved arbitrarily.

However, A* with tie-breaking strategy $[g + h^*, h^*]$ may expand more states than another tie-breaking strategy. Figure 4.1a shows an example with two paths to goal states using only zero-cost operators. The state s_0 is the initial state; doubly-circled states are goals and ellipses represent arbitrarily long transition sequences with cost zero. In this situation, $[g + h^*, h^*]$ provides no information. To reach a goal from s_0, A* may expand three states using the left s-path ($s_0 \to A \to C \to F$), or an arbitrarily large set of states using the right s-path ($s_0 \to B \to \cdots \to G \to H$). E_i represents a large set of states with f-value equal to 0.

4.2.2 Analyzing $\hat{h}^*$ as Tie-Breaking Strategy

Asai and Fukunaga (2017) have proposed to use *distance-to-go* heuristics to improve the performance of tie-breaking strategies. Distance-to-go heuristics ignore the costs of operators and just count the number of operators needed to reach a goal state. We denote distance-to-go heuristics as $\hat{h}$.

Definition 4.1 (Distance-to-go heuristic)**.** *Let $\mathcal{S} = \langle s_0, S_*, \mathcal{O}, cost\rangle$ be a state space and h be a heuristic for $\mathcal{S}$. A* distance-to-go heuristic *$\hat{h}$ is a heuristic function for $\mathcal{S}$, where for all $o \in \mathcal{O}$ the heuristic maps it to a new operator $\hat{o} \in \hat{\mathcal{O}}$ with $cost(\hat{o}) = 1$ and computes the heuristic function by replacing $\mathcal{O}$ by $\hat{\mathcal{O}}$ on $\mathcal{S}$.*

In other words, the distance-to-go heuristic $\hat{h}$ uses the same algorithm as h but replaces the cost of all operators by one. Thus $\hat{h}^*$ counts the exact number of operators necessary to reach the closest goal state for each state. Strategy $[g + h^*, \hat{h}^*]$ improves coverage in zero-cost domains (ASAI; FUKUNAGA, 2017).

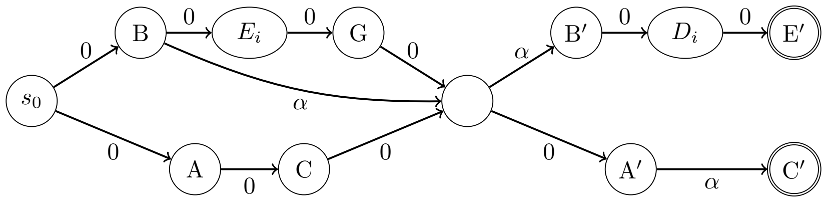
However, strategy $[g + h^*, \hat{h}^*]$ can also fail to produce an optimal expansion, as the example of Figure 4.1a shows. Let $\alpha > 0$. After expanding s_0, we have $\hat{h}^*(A) = 2$, because A can reach the closest goal F applying two operators, and $\hat{h}^*(B) = 1$, because B can reach its closest goal D applying only one operator. As a consequence, A* expands state B first. However, the s-path $s_0 \to B \to D$ is not optimal because the operator that enables B to reach goal state F has cost α. Thus A* expands four states ($\langle s_0, B, A, C\rangle$), and the optimal strategy only three ($\langle s_0, A, C\rangle$).

4.2.3 Analyzing h_c^* as Tie-Breaking Strategy

The perfect heuristic h^* guides the search through an cost-optimal path but fails to identify the cost-optimal path with the least number of operators; the distance-to-go heuristic $\hat{h}^*$ guides the search to expand a path with fewest operators to the goal but fails to estimate the total cost of the path. We can combine both estimates to improve the search performance.

Definition 4.2 (Cost-adapted heuristic)**.** *Let $\mathcal{S} = \langle s_0, S_*, \mathcal{O}, cost\rangle$ be a state space and h be a heuristic for $\mathcal{S}$. A* cost adapted heuristic *h_c is a heuristic function for $\mathcal{S}$, where for all $o \in \mathcal{O}$ the heuristic maps it to a new operator $o_c \in \mathcal{O}_c$ with $cost(o_c) = cost(o) + c$ and computes the heuristic function by replacing $\mathcal{O}$ by $\mathcal{O}_c$ on $\mathcal{S}$.*

Figure 4.2: Example of a state space where tie-breaking strategies $[g + h^*, h_c^*]$ using cost adaptation fail.



In other words, the cost adapted heuristic h_c computes the same heuristic function h on $\mathcal{S}$, but it adds a constant c to each operator cost. We will also say that a tie-breaking strategy based on h_c is a method *using cost adaptation*.

Richter, Westphal and Helmert (2011) introduced the idea of summing one to every operator cost in the LAMA solver. The intuition is that by doing so, A* can combine the operator cost with the cost of applying an operator. LAMA was a satisfiability solver and this cost adaptation could lead to a worse solution quality. However, using it as tie-breaker cannot change the solution quality. In the special case used in the LAMA solver with $c = 1$ we denote h_c as h_{+1}.

Now, we analyze the behavior of h_c for different magnitudes of c. First, consider $c = \epsilon$ where ϵ is a small constant such that $\epsilon \ll \min_{o \in \mathcal{O}}\{\text{cost}(o) \mid \text{cost}(o) > 0\}$. The effect of making ϵ very small is that even for the longest path of length l, the product $l\epsilon$ is still smaller than the least nonzero operator cost. If we apply $[g + h^*, h_\epsilon^*]$ to the example of Figure 4.1a it produces the optimal expansion $\langle s_0, A, C\rangle$.

However, $[g + h^*, h_\epsilon^*]$ can also fail. Figure 4.1b shows an example where A* with $[g+h^*, h_\epsilon^*]$ expands three states and the optimal expansion only two. In this example, after expanding s_0, A* can expand A and B, where $h_\epsilon^*(A) = \epsilon + \alpha$ while $h_\epsilon^*(B) = 2\epsilon + |D_i|\epsilon$. Thus, B is chosen for expansion, and then the states in D_i, leading to goal state E. A* expands the path $s_0 \to B \to \cdots \to E$ instead of the shortest s-path $s_0 \to A \to C$.

An approach to solve the example of Figure 4.1b is to use $c = \mathcal{M}$, where $\mathcal{M} \gg \max_{o \in \mathcal{O}}(\text{cost}(o))$. In Figure 4.1b breaking ties by $h_\mathcal{M}^*$ produces the optimal expansion. Now, $h_\mathcal{M}^*(A) = \alpha + \mathcal{M}$ and $h_\mathcal{M}^*(B) = 2\mathcal{M} + |D_i|\mathcal{M}$. Since $\mathcal{M} \gg \alpha$, A* expands A instead of B, followed by the expansion of C, leading the search to the optimal expansion sequence $\langle s_0, A\rangle$.

However, $h_\mathcal{M}^*$ fails to achieve the optimal expansion in the example of Figure 4.1a, where we have $h_\mathcal{M}^*(A) = 2\mathcal{M}$ and $h_\mathcal{M}^*(B) = \mathcal{M} + \alpha$. Since $\alpha \ll \mathcal{M}$, we have $h_\mathcal{M}^*(A) > h_\mathcal{M}^*(B)$ causing the search to expand B, leading to the same problem from last subsection.

One may wonder if there is a value for c that works "universally" for any task. Theorem 4.1 proves this is not the case.

Theorem 4.1. *There is no c, such that for any $\mathcal{S}$, the A^* algorithm with $[g + h^*, h^*_c]$ guarantees an optimal expansion.*

Proof. In the example of Figure 4.1a, we obtain the optimal expansion iff $c < \alpha$. In the example of Figure 4.1b, when $|D_i| = 1$, we obtain the optimal expansion iff c is sufficiently large. If we join the examples by identifying goal states F, D, and H in Figure 4.1a and the initial state s_0 in Figure 4.1b by merging them into a single normal state, we obtain the instance shown in Figure 4.2 that has to satisfy both conditions for an optimal expansion. Hence, it is impossible to guarantee an optimal expansion using only cost adaptation. $\square$

Despite this, cost adaptation will be useful to find an optimal expansion tie-breaking strategy later on.

The cost adapted heuristic h_c has two corner cases. For $c = 0$ we obtain $h_0 = h$. For $c \to \infty$ tie-breaker h^*_c leads to the same expansion sequence as $\hat{h}$, because a sufficiently large cost c will dominate the operator costs, and the shortest path will be also the cheapest.

4.2.4 Special Cases

We now explain two special cases that admit optimal tie-breaking strategies using only the techniques presented above. These special cases have restrictions on the operator costs, but are common in practice. First, we define a *improving successor* state:

Definition 4.3 (Improving Successor). *A state s' is an* improving successor *of state s, if s' is a successor with $h^*(s') < h^*(s)$.*

In the case where domains have no zero-cost operators, we can state the following:

Lemma 4.1. *Given h^* for a state space $\mathcal{S}$ where cost$(o) > 0$ for all $o \in \mathcal{O}$, A^* with tie-breaking strategy $[g + h^*, h^*]$ will expand only states on one cost-optimal path.*

Proof. We will show that A* with h^* will expand a sequence of improving successors along one cost-optimal path.

Consider the case where $\mathcal{S}$ is solvable. After expanding s_0, A* will select for expansion a state $s = \arg\min_{s \in succ(s_0)} h^*(s)$ with $f(s) = \mathcal{C}^*$. Thus, the next expanded state is an improving successor of s_0.

Let s be any state expanded during the search. Note that s is solvable since there will always exist a solvable state in OPEN and it would be preferable for expansion than any other unsolvable state.

Since s is a solvable state with $f(s) = \mathcal{C}^*$ and cost$(o) > 0$ for all $o \in \mathcal{O}$. Then, there exists at least one improving successor $s' = \arg\min_{s' \in succ(s)} h^*(s')$ with $f(s') = \mathcal{C}^*$. Such state s' will have minimum h-value among all state in OPEN. Thus, the next expanded state s' a direct improving successor of s and by the induction hypothesis s a direct improving successor of the previously expanded state. Thus, all expanded states are part of one cost-optimal path.

It remains to be shown that A* terminates. Since cost$(o) > 0$ for all $o \in \mathcal{O}$, at every iteration of A* the minimum h^*-value in OPEN is decreased and eventually a goal state with $h^*(s_*) = 0$ will be inserted in OPEN. At that point the goal state is removed from open and the search terminates.

In the case $\mathcal{S}$ is unsolvable A* with h^* will terminate the search before the beginning of the search. □

In state spaces where all operators have the same strictly positive cost, we can guarantee optimal expansion by using tie-breaking strategy $[g + h^*, h^*]$.

Corollary 4.1. *Given h^* for a state space $\mathcal{S}$ where cost$(o) = \alpha > 0$ for all $o \in \mathcal{O}$, A* with tie-breaking strategy $[g + h^*, h^*]$ will have optimal expansion.*

Proof. By Lemma 4.1, A* will expand only states along one optimal path. In the case of state spaces with uniform costs cost$(o) = \alpha > 0$, $\mathcal{C}^* = h^*(s_0) = n\alpha$, where n is the number of operators in any cost-optimal path. Thus, any path expanded by A* has minimum number of states. □

In the case where all costs are 0, we know that all states will have $h^* = 0$. The strategy $[g + h^*, h^*_c]$ with $c > 0$, leads the search to an optimal expansion.

Corollary 4.2. *Given h^* for a state space $\mathcal{S}$ where cost$(o) = 0$ for all $o \in \mathcal{O}$, A* with tie-breaking strategy $[g + h^*, h^*_c]$ for any $c > 0$ will have optimal expansion.*

Proof. Using h_c, we will modify the cost function of $\mathcal{S}$ for all operators in $\mathcal{O}$. Adding a constant $c > 0$ to all costs will lead us to a new state space with uniform costs greater than zero. Hence, our Corollary 4.1 applies. □

4.2.5 An Optimal Expansion Strategy based on h^*_c

In this subsection, we present the tie-breaking strategy $g + h^*_c$ and prove that A* with $[g + h^*, g + h^*_\epsilon]$ guarantees optimal expansion in any instance, given h^*. Given the definitions and results from the previous section, we can state the following:

Theorem 4.2. *Given h^* for a state space $\mathcal{S}$ where cost$(o) > 0$ for all $o \in \mathcal{O}$, A* with tie-breaking strategy $[g+h^*, g+h^*_\epsilon]$ will expand only states on one cost-optimal path with the least number of operators.*

Proof. After s_0 is expanded, A* will select for expansion one state $s \in succ(s_0)$ with $f^*(s) = \mathcal{C}^*$ and $s = \arg\min_{\forall s \in succ(s_0)} g(s) + h^*_\epsilon(s)$ – i.e., one state along a cost-optimal path ($g + h^*$) with the least number of operators ($h^*_\epsilon(s) - h^*(s)$). By Lemma 4.1 we know that A* will expand only one such path and hence A* with $[g + h^*, g + h^*_\epsilon]$ has optimal expansion. □

Theorem 4.2 only holds for state spaces where all operator costs are strictly positive, but it is straight-forward to generalize it to cases where zero-cost operators are also allowed. Thus, the corollary below follows:

Corollary 4.3. *Given h^* for a state space $\mathcal{S}$, A* with tie-breaking strategy $[g+h^*, g+h^*_\epsilon]$ will expand only states on one cost-optimal path with the least number of operators.*

Proof. Only states s with $f(s) = \mathcal{C}^*$ will be expanded. Since all $s \notin S_*$, $h^*_\epsilon(s) > 0$ there will always be a improving successor for s in the tie-breaking strategy. Thus, A* with $[g + h^*, g + h^*_\epsilon]$ has optimal expansion. □

Tie breaking by $g + h^*_\epsilon$ is similar to adding a small and constant noise to the operator costs. This noise, however, is sufficient to discriminate between optimal paths with a different number of expansions. When the noise of all operators on a path is summed, A* can identify which of the candidate states will lead to an optimal expansion.

4.2.6 Analyzing Approximations of h^*

In this subsection, we discuss tie-breakers in the context of h as an admissible approximation of h^*. In this setting, we show that $[g + h^*, g + h_\epsilon]$ does not guarantee an optimal expansion but the strategy $[g + h, g + h^*_\epsilon]$ does.

To show that an arbitrary approximation h can expand states that do not belong to the cost-optimal path with the least number of operators, consider the example of Figure 4.1a and assume that α is the cost of operator o_α and $\alpha \gg 1$. Let h' be an approximation of h^* that is incapable of capturing the necessity of applying operator o_α – i.e., it considers the cost of operator o_α to be 0. Since our first evaluation still uses h^*, we have $f(A) = f(B) = \mathcal{C}^* = 0$ for the successors A and B of s_0. To break this tie, we use $g + h_\epsilon$. We have $g + h_\epsilon(A) = 2\epsilon$ and $g + h_\epsilon(B) = \epsilon$ due to the possible path $s_0 \to B \to D$ where h cannot predict the need of o_α. Hence, B is expanded instead of A, failing to expand only the cost-optimal path with the least number of operators.

Dechter and Pearl (1985) proved that there is always a tie-breaking strategy τ for any pathological state space topology $\mathcal{T} = \langle \mathcal{S}, h\rangle$ such that A* with $[g + h, \tau]$ presents optimal expansion. Indeed, if h is admissible and consistent, we can state that $[g + h, g + h^*_\epsilon]$ presents optimal expansion. The proof is similar to Theorem 4.2 and Corollary 4.3, however, we need to consider the subset of states $E \subseteq \mathcal{S}$ representing the *entry states* of the final f-layer with $f(s) = \mathcal{C}^*$ for every $s \in E$ – i.e., states not yet expanded with f-value equals to $\mathcal{C}^*$.

Theorem 4.3. *Given an admissible and consistent heuristic function h and h^* for a state space $\mathcal{S}$, A* with tie-breaking strategy $[g + h, g + h^*_\epsilon]$ will present optimal expansion.*

Proof. During the search, any tie-breaking strategy cannot avoid any expansion for states s which $f(s) < \mathcal{C}^*$. Hence, we have to guarantee to minimize the number of expansions in the last f-layer by always selecting the best choice among the states in E.

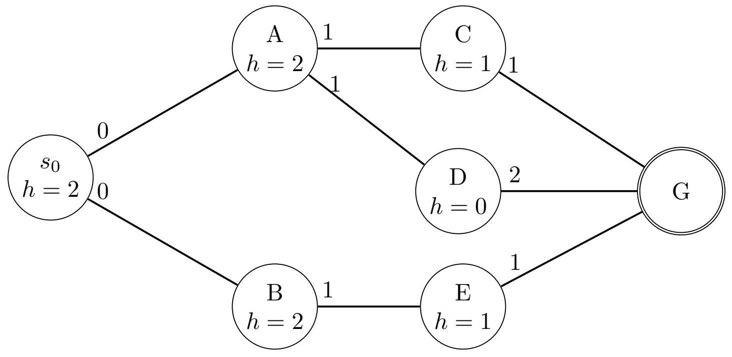
Once we do not have any state s with $f(s) < \mathcal{C}^*$ in OPEN, we select the states in E accordingly to $[g + h, g + h^*_\epsilon]$. Since our heuristic function h is consistent, at this point, we will never generate a successor state s with $f(s) < \mathcal{C}^*$. Therefore, we can consider that our search now is similar to a search using h^*.

Due to admissibility, there are only two cases in E: $i)$ states where the function f is underestimated; $ii)$ states in which the f-value is perfect. Our tie-breaking strategy will select the state $s = \arg\min_{s \in E}\ (g(s) + h^*(s) + d(s)\epsilon)$ where $d(s)$ is the number of operators from state s to the closest goal state through an optimal path[2]. Hence, states in the former case will never be expanded by our tie-breaking strategy because their value for $g + h^*$ will always be greater than $g + h^* + d\epsilon$ for some other state in E.

In the latter case, all states in E will present $g + h^*$ constant; consequently, we

[2]Here, we are using the fact that $g + h^*_\epsilon = g + h^* + d\epsilon$, where d is the number of operators necessary to reach the goal.

Figure 4.3: Example of a state space topology where tie-breaking strategy $[g+h, g+h^*_\epsilon]$ fails if h is inconsistent.



will select the state $s = \arg\min_{s\in E, f^*(s)=f(s)} d(s)\epsilon$. Since our tie-breaker now is only dependent on function d, there will always be at least one improving successor in $\text{succ}(s)$ that will be placed in E. By definition, this improving successor s' will have $d(s')\epsilon < d(s)\epsilon$. Thus, in the next selection this improving successor will be expanded accordingly to our selection function $\arg\min_{s\in E, f^*(s)=f(s)} d(s)\epsilon$ (if there is more than one improving successor with same d, we break ties arbitrarily).

This behavior will follow successively until we find a goal state. At this point, our search will finish and it expanded only the states where $f < \mathcal{C}^*$ plus the optimal path with the least number of operators in E. □

Notice, however, that the plan found is not guaranteed to be the optimal plan with the least number of operators. It is only guaranteed to be the optimal plan with least number of operators in the last f-layer, because this is the only layer where we are able to save expansions.

Figure 4.3 illustrates a state space topology where tie-breaking strategy $[g+h, g+h^*_\epsilon]$ fails if h is inconsistent. Heuristic values are showed for each state in the topology. To achieve optimal expansion the algorithm should expand paths $s_o \rightarrow A \rightarrow C \rightarrow G$ or $s_o \rightarrow B \rightarrow E \rightarrow G$. However, whenever we expand state A, we *must* expand state D as well. Due to the inconsistency of the heuristic function h, we have $f(D) < f(A)$ and $\arg\min_{s\in \textsf{OPEN}} f(s) = D$, hence this successor must be expanded before than any other successor of A. Since our tie-breaking strategy $[g+h, g+h^*_\epsilon]$ cannot guarantee to favor the expansion of B over the expansion of A, it does not guarantee optimal expansion if h is inconsistent.

The result from Theorem 4.3 is useful from two perspectives: (i) if we have a consistent but inadmissible heuristic h where $h = h^*$ in a significant number of states but $h > h^*$ in only a few, we cannot use it as our main heuristic to guide A*, however we can use it to guide our tie-breaking strategy and let an admissible (but not so informed) heuristic in our main evaluation function; (ii) in the case where h is really expensive to compute, we could tweak our A* to only compute our tie-breakers in f-plateaus that may have goal states (we would also need to try to predict if an f-plateau contain such a state) and guide our search only by using a heuristic that is faster to compute in the other plateaus.

4.3 Experiments

Table 4.1: Comparison of the geometric mean of the number of expanded states using different methods for heuristic function and tie-breaking strategy.

	h^*, h^*		h^*, h^η		h^η, h^*		h^η, h^η		h^η, h^{FF}	
$\eta =$ **LM-cut**	IPC	Z	IPC	Z	IPC	Z	IPC	Z	IPC	Z
$[g+h^1, h^2]$	12.05	124.49	13.34	244.44	69.92	549.94	79.24	805.68	80.79	690.19
$[g+h^1, \hat{h}^2]$	**11.78**	13.33	14.28	23.72	69.87	119.02	79.29	172.46	80.88	156.04
$[g+h^1, h^2_{+1}]$	**11.78**	13.37	13.01	**20.18**	69.87	105.57	**79.18**	147.79	80.95	**131.62**
$[g+h^1, h^2_\epsilon]$	**11.78**	13.39	**12.63**	21.33	69.88	105.57	79.36	**144.93**	**79.63**	142.02
$[g+h^1, g+h^2_\epsilon]$	**11.78**	**13.26**	31.91	65.67	**69.84**	**104.67**	80.71	145.88	81.26	141.08
$\eta =$ **iPDB**										
$[g+h^1, h^2]$			13.41	249.71	102.61	635.42	119.62	1226.78	115.59	851.96
$[g+h^1, \hat{h}^2]$			13.53	19.63	102.61	172.10	**119.49**	217.62	115.59	206.25
$[g+h^1, h^2_{+1}]$			13.04	**18.83**	102.61	122.10	119.51	**180.30**	115.59	**160.20**
$[g+h^1, h^2_\epsilon]$			**13.03**	19.97	102.61	122.28	119.60	199.37	**113.84**	164.38
$[g+h^1, g+h^2_\epsilon]$			32.12	68.26	**102.40**	**121.11**	120.35	227.35	132.45	204.17
$\eta =$ **M&S**										
$[g+h^1, h^2]$			13.51	178.94	80.67	282.56	85.30	319.80	87.24	357.37
$[g+h^1, \hat{h}^2]$			**12.91**	20.18	80.66	45.10	**84.95**	67.92	87.50	61.03
$[g+h^1, h^2_{+1}]$			13.90	**18.13**	80.67	44.51	85.03	60.79	87.49	**54.97**
$[g+h^1, h^2_\epsilon]$			13.77	18.72	80.67	44.56	85.06	**60.07**	**86.74**	58.22
$[g+h^1, g+h^2_\epsilon]$			26.19	42.24	**80.23**	**44.06**	85.17	66.18	107.05	86.23

In our experiments, we tested the improvement of state expansions, search time and coverage for the different methods studied here and previously mentioned in the literature. The experiments use revision 6251 of the Fast-Downward planning system (HELMERT, 2006a) with the modifications of Asai and Fukunaga (2017). We use the same benchmarks as Asai and Fukunaga (2017). In total, we used 1104 instances from IPC and 620 from their zero-cost benchmarks. All experiments have been run on a PC with an AMD FX-8150 processor running at 3.6 GHz and 32 GB of main memory. When using some heuristic h as a tie-breaker any remaining ties are broken by FIFO order.

4.3.1 Comparing Theory and Practice

We first focus on the question if the theoretical advantage of cost adaptation strategies translates into practice. For these experiments we use a time limit of 30 minutes and a memory limit of 4 GB, and a subset of 183 IPC and 87 zero-cost instances, which could be solved optimally by all methods given these limits and the internal limits of Fast-Downward to build the perfect heuristic h^*. Thus, this reduced set of benchmarks contains instances with smaller state spaces than usual.

Table 4.1 reports the geometric mean of the number of expanded states for different combinations of the primary A* heuristic and tie-breaker. For each combination, the Table shows the results for IPC and zero-cost instances separately. The pair at the header of each column is denoted by h^1, h^2, where h^1 was used as the heuristic for the evaluation function and h^2 as the heuristic for the tie-breaking strategy. Each one of the three parts of the table presents results for a different heuristic h^η where η is specified in each part. The best results for each column in each part is painted.

We first analyze the theoretical predictions using h^* as the heuristic function and tie-breaker in the first two columns of Table 4.1. Based on our analysis breaking ties with $g + h^*_\epsilon$ must expand the least number of states, and we can see that this is indeed the case. We can also see that using the reduced benchmarks with a small state space, A* expands very few states. In spite of this, breaking ties with $g + h^*_\epsilon$ performs best, and strictly dominates the other tie-breakers on zero-cost instances.

In the second combination we relax the tie-breaker to h^η and we test for different heuristics η. As it can be expected, the number of expanded states increases for all tie-breakers. The theoretical results do not guarantee an optimal performance of $g+h^*_\epsilon$ in this case, and indeed we can see that it actually performs worse that other strategies. This can possibly be explained by the fact the $h^{\text{LM-cut}}$, h^{iPDB} and $h^{\text{M\&S}}$ are not fully informed. Thus, when a successor state on an cost-optimal path is generated it tends to have a higher value of $g + h^*_\epsilon$, and leads A* to first expand less informed states. This effect is less pronounced for tie-breakers not using g.

In the remaining combinations, we switch roles and focus on not fully informed searches using heuristic h^η with different tie breakers. In all these cases, A* expands a significantly higher number of states. When breaking ties with h^* (fifth and sixth columns in Table 4.1), all cost adaptation methods have a similar performance on the IPC instances, and breaking ties by $g + h^*_\epsilon$ is the best method.

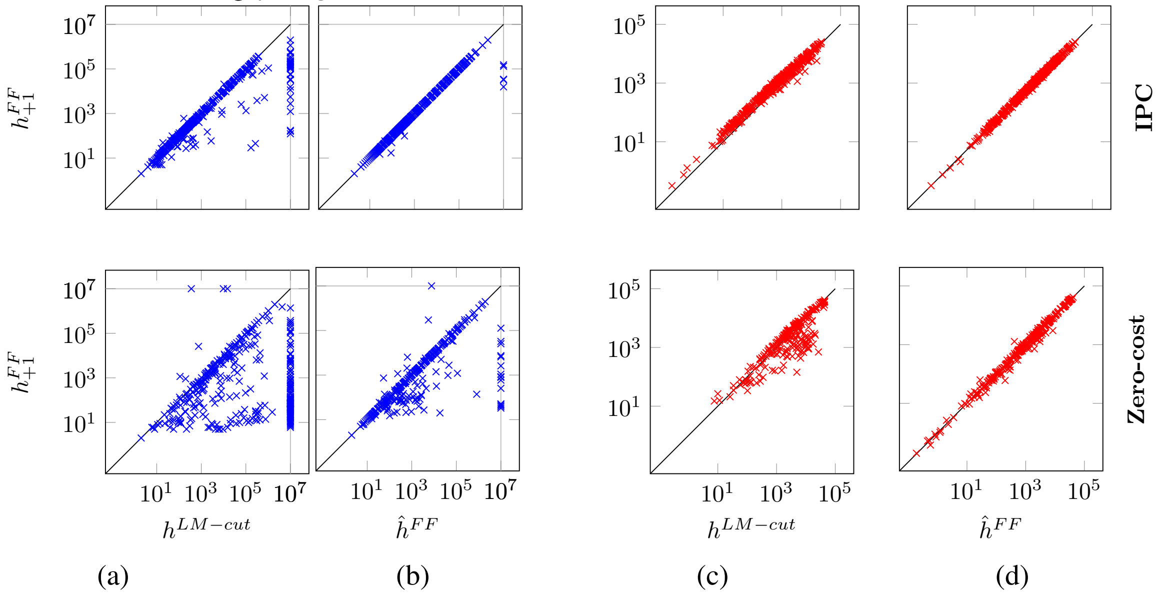
We finally relax the tie-breaker to h^η and h^{FF} (HOFFMANN; NEBEL, 2001). We have selected heuristic h^{FF} as Asai and Fukunaga (2017). Note that heuristic h^{FF} is not admissible, but will not change the optimality of the search when used as a tie-breaker. Both cases expand more states than the optimal tie-breaker (as expected), but the relative performance of the tie-breakers is very similar, with little difference on the IPC. On the zero-cost instances, breaking ties by $\hat{h}$ is always worst, and methods using cost adaption are always the best.

In summary, all cost adaptation strategies are similar on the IPC instances, but far better than the default tie-breaker h on zero-cost. Our results show that even in small state spaces and using the perfect heuristic h^*, tie-breakers are important, even when not optimal. Still, the heuristic function is more important than the tie-breaker, as the comparison between the second and the third combinations confirms. The last two combinations show that tie-breakers also make a difference in practice, and there is enough room for improvement.

Table 4.2: Comparison of the number of solved instances in IPC and zero-cost benchmarks where the value f is $g + h^{\text{LM-cut}}$.

Method	**IPC** (1104)	**Zero-cost** (620)
$[f, h^{\text{LM-cut}}]$	525	237
$[f, \hat{h}^{\text{LM-cut}}]$	531	301
$[f, h_{+1}^{\text{LM-cut}}]$	530	299
$[f, h_{\epsilon}^{\text{LM-cut}}]$	532	301
$[f, g + \hat{h}^{\text{LM-cut}}]$	417	288
$[f, g + h_{+1}^{\text{LM-cut}}]$	531	299
$[f, g + h_{\epsilon}^{\text{LM-cut}}]$	524	300
$[f, h^{\text{FF}}]$	548	251
$[f, \hat{h}^{\text{FF}}]$	557	338
$[f, h_{+1}^{\text{FF}}]$	**562**	**352**
$[f, h_{\epsilon}^{\text{FF}}]$	559	351
$[f, g + \hat{h}^{\text{FF}}]$	457	323
$[f, g + h_{+1}^{\text{FF}}]$	558	350
$[f, g + h_{\epsilon}^{\text{FF}}]$	553	346
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \text{LIFO}]$	530	328

Figure 4.4: Expanded states and expansions per second for tasks in IPC (top) and Zero-cost (bottom) using $f = g + h^{\text{LM-cut}}$ with different tie-breakers (axis).



4.3.2 Performance on the Complete Set of Instances

We now turn to the practical performance of tie-breakers using cost adaptation. Our second experiment compares the coverage of different tie-breaking strategies using $f = g + h^{\text{LM-cut}}$ to guide the search in the complete set of 1104 IPC and 620 zero-cost instances. In this experiment we have imposed limits of 4 GB and 5 minutes for each run, following Asai and Fukunaga (2017).

The results are shown in Table 4.2. We compare our main cost adaptation methods against the standard methods in the literature and the current best deterministic tie-breaker from Asai and Fukunaga (2017). (The best non-deterministic tie-breaker of Asai and Fukunaga (2017) solves in average 2.3 instances more.) Highlighted in blue are the best results for each column; in greyare the results for the best method of Asai and Fukunaga (2017) for the IPC instances (upper entry) and zero-cost instances (lower entry). Looking at the group of tie-breakers using $h^{\text{LM-cut}}$ we find that that all methods using cost adaption perform better than the standard tie-breaker h. Also, methods using only cost adaptation dominate methods adding g, but the theoretically best tie breaker $g + h_{\epsilon}^{\text{LM-cut}}$ for h^* is only slightly worse.

The second group using h^{FF} dominates the strategies using $h^{\text{LM-cut}}$ only. This confirms the observation of Asai and Fukunaga (2017) that breaking ties by h^{FF} is better than $h^{\text{LM-cut}}$. However, we find that $\hat{h}^{\text{FF}}$ also performs better on zero-cost instances than their best strategy. This can probably be explained by the difference between processor speeds.

Again the tie-breaker $g + h^{\text{FF}}_{\epsilon}$ which is theoretically best for h^* is competitive. The overall best method is h^{FF}_{+1}. It solves five instances more on the IPC benchmark than $\hat{h}^{\text{FF}}$, the best tie-breaker from the literature. The best known tie-breaker for zero-cost instances is $[f, \hat{h}^{\text{FF}}, \langle d \rangle$, LIFO]. Here h^{FF}_{+1} solves 24 instances more.

Figures 4.4a and 4.4b compare the number of expanded states of the best method $[f, h^{\text{FF}}_{+1}]$ against the most used method in literature, $[f, h^{\text{LM-cut}}]$ and the best method from the literature $[g + h, \hat{h}^{\text{FF}}]$. The plots on top show results for IPC instances, the ones on the bottom for zero-cost.

We see that tie-breaking with h^{FF}_{+1} expands fewer states on most of the instances compared to $h^{\text{LM-cut}}$, in particular on the zero-cost instances. The number of expanded states compared to $\hat{h}^{\text{FF}}$ is similar in IPC. In zero-cost instances h^{FF}_{+1} outperforms $\hat{h}^{\text{FF}}$, expanding fewer states in general.

Another important issue about tie-breaking strategies is the overhead to compute a second evaluation function. To analyze this issue, Figures 4.4c and 4.4d compare the expansions per second of the methods. We find that all methods expand about the same number of states per second, with the exception of $h^{\text{LM-cut}}$ on zero-cost instances.

In general lines, the “pure” cost adaptation methods ($[f, h_c]$) using the h^{FF} heuristic have the best performance. Tie-breaking by h^{FF}_{+1} presents the best coverage in both benchmarks.

4.4 Conclusion and Future Work

In this chapter, we presented a tie-breaking strategy for A* with h^* that guarantees the minimum number of expanded states among all tie-breaking strategies. Our analysis showed that even for the perfect heuristic h^*, where A* only expands states s with $f(s) = \mathcal{C}^*$, previously proposed tie-breakers fail in producing an optimal A*. Our results, based on the analysis of Dechter and Pearl (1985), showed how to build an optimal A* for h^*.

Our experiments confirm the results from Asai and Fukunaga (2017) that tie-breakers have the potential to increase coverage and reduce the number of expanded states. In a nutshell, our best method based on h_c solves 152 instances more than h, the most common tie-breaker in the literature, and 19 more than Asai and Fukunaga (2017)’s best method, the state of the art in the literature. Our experiments showed that even in small state spaces and with the perfect heuristic h^*, the performance of A* can be improved by a better tie-breaking strategy. Our main contribution in this work is to provide an analysis that enables a better understanding of the role of tie-breaking strategies in the performance of A*.

Three main ideas may be interesting to investigate further. The first is an analysis similar to the one by Helmert and Röger (2008) which investigated for specific domains the performance of A* with almost perfect heuristics, one could do the same with almost perfect tie-breakers. Second, one may study the existence of effective domain-dependent tie-breakers, not based on h^*.

5 THESIS' CONCLUSION

In this thesis, we studied the difference in performance between simple domain-dependent and state-of-the-art heuristics in one of the hardest domains in IPC. Also, we proposed a novel method of tie-breakers and proved that this method is an optimal tie-breaker.

In the former part, one of our intentions was to compare the coverage between domain-independent and domain-dependent heuristics and see how is the performance of domain-independent heuristics in literature nowadays compared to simple domain-dependent methods. To do so, we decided to study especifically the airport ground traffic problem, since it is has a practical aspect in the real world and it is considered one of the hardest domains in IPC. We proposed and tested two simple heuristics based only on the distances from each airplane to a goal segment (closest goal and matching heuristic). We also modified the IPC original domain to allow free parking positions to each plane. Hence, we could verify the performance of the studied heuristics in a domain that is harder and closer to reality. Our experiments in the original and the new version of the domain showed that our proposed heuristics can obtain better coverage and be much faster than the state-of-the-art heuristics in classical planning literature.

Our latter study analyzed theoretically some tie-breaking strategies in the perfect scenario – i.e., state spaces $\mathcal{S}$ in which we have h^*. We showed that breaking ties in favor of just h or $\hat{h}$ can present flaws and also proposed a first method based on cost adaptation similar to the LAMA solver. Therefore, we proved that tie-breaking strategy $g+h^*_\epsilon$ always achieves optimal expansion. Furthermore, we tested all the tie-breaking strategies in IPC and zero-cost domains and showed that, indeed, a change in tie-breakers can increase coverage and state expansion significantly.

As a last point, we state that the results presented in this work are not only valuable for the automated planning community, but also for anyone interested in search or general problem solving.

REFERENCES

ASAI, M.; FUKUNAGA, A. Tie-breaking strategies for cost-optimal best first search. **Journal of Artificial Intelligence Research**, v. 58, p. 67–121, 2017.

ASAI, M.; FUKUNAGA, A. S. Tiebreaking strategies for A^* search: How to explore the final frontier. In: **AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2016. p. 673–679.

BÄCKSTRÖM, C.; NEBEL, B. Complexity results for SAS+ planning. **Computational Intelligence**, Wiley Online Library, v. 11, n. 4, p. 625–655, 1995.

BONET, B. An admissible heuristic for SAS+ planning obtained from the state equation. In: **International Joint Conference on Artificial Intelligence**. [S.l.: s.n.], 2013. p. 2268–2274.

BONET, B.; GEFFNER, H. Planning as heuristic search. **Artificial Intelligence**, Elsevier, v. 129, n. 1-2, p. 5–33, 2001.

BORRAJO, D. et al. (Ed.). **Symposium on Combinatorial Search**. [S.l.]: AAAI Press, 2012. ISBN 978-1-57735-584-7.

BRIEL, M. V. D. et al. An LP-based heuristic for optimal planning. **Principles and Practice of Constraint Programming–CP 2007**, Springer, p. 651–665, 2007.

BYLANDER, T. The computational complexity of propositional strips planning. **Artificial Intelligence**, Elsevier, v. 69, n. 1-2, p. 165–204, 1994.

CORRÊA, A. B.; PEREIRA, A. G.; RITT, M. Improved airport ground traffic control with domain-dependent heuristics. In: IEEE. **Brazilian Conference on Intelligent Systems**. [S.l.], 2016. p. 73–78.

CULBERSON, J. C.; SCHAEFFER, J. Pattern databases. **Computational Intelligence**, Wiley Online Library, v. 14, n. 3, p. 318–334, 1998.

DECHTER, R.; PEARL, J. Generalized best-first search strategies and the optimality of a. **Journal of the ACM**, ACM, v. 32, n. 3, p. 505–536, 1985.

DORAN, J. E.; MICHIE, D. Experiments with the graph traverser program. In: THE ROYAL SOCIETY. **Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences**. [S.l.], 1966. v. 294, n. 1437, p. 235–259.

EDELKAMP, S. Planning with pattern databases. In: **European Conference on Planning**. [S.l.: s.n.], 2001. p. 13–24.

EDELKAMP, S. Automated Creation of Pattern Database Search Heuristics. In: **Workshop on Model Checking and Artificial Intelligence**. [S.l.: s.n.], 2007. p. 35–50.

FAN, G.; MÜLLER, M.; HOLTE, R. The two-edged nature of diverse action costs. In: **International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2017.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE transactions on Systems Science and Cybernetics**, IEEE, v. 4, n. 2, p. 100–107, 1968.

HASLMUM, P.; GEFFNER, H. Admissible heuristics for optimal planning. In: **International Conferene of AI Planning Systems**. [S.l.: s.n.], 2000. p. 140–149.

HASLUM, P. et al. Domain-independent construction of pattern database heuristics for cost-optimal planning. In: **AAAI Conference on Artificial Intelligene**. [S.l.: s.n.], 2007. p. 1007–1012.

HATZACK, W. **Entwicklung und Auswertung von Algorithmen zur autonomen Verkehrskoordinierung und Konfliktauflösung an Flughäfen**. Thesis (PhD) — University of Freiburg, 2002.

HATZACK, W.; NEBEL, B. The operational traffic control problem: Computational complexity and solutions. In: **Rec. Adv. AI Planning, 6h Europ. Conf. Planning**. [S.l.: s.n.], 2001.

HELMERT, M. On the complexity of planning in transportation and manipulation domains. **Master's thesis, Albert-Ludwigs-Universit at Freiburg**, 2001.

HELMERT, M. The fast downward planning system. **Journal of Artificial Intelligence Research**, v. 26, p. 191–246, 2006.

HELMERT, M. New complexity results for classical planning benchmarks. In: **International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2006.

HELMERT, M.; DOMSHLAK, C. Landmarks, critical paths and abstractions: What's the difference anyway? In: GEREVINI, A. et al. (Ed.). **International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2009. p. 162–169.

HELMERT, M.; HASLUM, P.; HOFFMANN, J. Flexible abstraction heuristics for optimal sequential planning. In: **International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2007. p. 176–183.

HELMERT, M. et al. Flexible abstraction heuristics for optimal sequential planning. In: **International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2007. p. 176–183.

HELMERT, M.; RÖGER, G. How good is almost perfect? In: **AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2008. v. 8, p. 944–949.

HEUSNER, M.; KELLER, T.; HELMERT, M. Understanding the search behaviour of greedy best-first search. In: **Symposium on Combinatorial Search**. [S.l.: s.n.], 2017.

HOFFMANN, J. et al. Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4. **Journal of Artificial Intelligence Research**, v. 26, p. 453–541, 2006.

HOFFMANN, J.; NEBEL, B. The FF planning system: Fast plan generation through heuristic search. **Journal of Artificial Intelligence Research**, v. 14, p. 253–302, 2001.

KARPAS, E.; DOMSHLAK, C. Cost-optimal planning with landmarks. In: **International Joint Conference on Artificial Intelligence**. [S.l.: s.n.], 2009. p. 1728–1733.

KATZ, M.; DOMSHLAK, C. Structural patterns heuristics via fork decomposition. In: **International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2008. p. 182–189.

KOLMOGOROV, V. Blossom V: a new implementation of a minimum cost perfect matching algorithm. **Mathematical Programming Computation**, Springer, v. 1, n. 1, p. 43–67, 2009.

PEREIRA, A. G.; RITT, M.; BURIOL, L. S. Finding optimal solutions to Sokoban using instance dependent pattern databases. In: HELMERT, M.; RÖGER, G. (Ed.). **Symposium on Combinatorial Search**. [S.l.]: AAAI Press, 2013. ISBN 978-1-57735-632-5.

PEREIRA, A. G.; RITT, M.; BURIOL, L. S. Solving Sokoban optimally using pattern databases for deadlock detection. In: **Anais do XI Encontro Nacional de Inteligência Artificial (ENIA)**. [S.l.: s.n.], 2014.

PEREIRA, A. G.; RITT, M.; BURIOL, L. S. Optimal Sokoban solving using pattern databases with specific domain knowledge. **Artificial Intelligence**, v. 227, p. 52–70, 2015.

POMMERENING, F. et al. From non-negative to general operator cost partitioning. In: **AAAI**. [S.l.: s.n.], 2015. p. 3335–3341.

PORTEOUS, J.; SEBASTIA, L.; HOFFMANN, J. On the extraction, ordering, and usage of landmarks in planning. In: **Sixth European Conference on Planning**. [S.l.: s.n.], 2014.

RICHTER, S.; HELMERT, M.; WESTPHAL, M. Landmarks revisited. In: **AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2008. v. 8, p. 975–982.

RICHTER, S.; WESTPHAL, M.; HELMERT, M. LAMA 2008 and 2011. In: **International Planning Competition**. [S.l.: s.n.], 2011. p. 117–124.

SHARON, G. et al. Meta-agent conflict-based search for optimal multi-agent path finding. In: BORRAJO, D. et al. (Ed.). **Symposium on Combinatorial Search**. [S.l.]: AAAI Press, 2012. p. 97–104. ISBN 978-1-57735-584-7.

SHARON, G. et al. The increasing cost tree search for optimal multi-agent pathfinding. In: **International Joint Conference on Artificial Intelligence**. [S.l.: s.n.], 2011. p. 662–667.

SIEVERS, S.; ORTLIEB, M.; HELMERT, M. Efficient implementation of pattern database heuristics for classical planning. In: BORRAJO, D. et al. (Ed.). **Symposium on Combinatorial Search**. [S.l.]: AAAI Press, 2012. p. 105–111. ISBN 978-1-57735-584-7.

STANDLEY, T. Finding optimal solutions to cooperative pathfinding problems. In: **AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2010. p. 173–178.

STANDLEY, T.; KORF, R. Complete algorithm for coorperative pathfinding problems. In: **International Joint Conference on Artificial Intelligence**. [S.l.: s.n.], 2011. p. 668–673.

TRüG, S.; HOFFMANN, J.; NEBEL, B. Applying automatic planning systems to airport ground-traffic control – a feasibility study. In: **Annual German Conference on Artificial Intelligence (KI)**. [S.l.: s.n.], 2004. (LNCS, 3238), p. 183–197.

WILDE, B. de; MORS, A. W. ter; WITTEVEEN, C. Push and rotate: a complete multi-agent pathfinding algorithm. **Journal of Artificial Intelligence Research**, v. 51, p. 443–492, 2014.