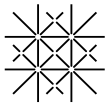


University of Basel

Potential Heuristics for Satisficing Planning

Master's Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence
http://ai.cs.unibas.ch

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. Silvan Sievers

Alexander Rovner
alexander.rovner@unibas.ch
2015-050-289

February 4, 2020

Acknowledgments

I would like to thank Prof. Dr. Malte Helmert for the opportunity to write this thesis as well as Dr. Silvan Sievers for supervising the project, providing crucial insight, and always finding the time to give feedback. Furthermore, I want to thank my friends and family for the mental support throughout the entirety of my studies. Special thanks go to Augusto B. Corrêa and Simon Dold for all the spontaneous brainstorming sessions and discussions.

Abstract

Potential heuristics are a class of heuristics used in classical planning to guide a search algorithm towards a goal state. Most of the existing research on potential heuristics is focused on finding heuristics that are admissible, such that they can be used by an algorithm such as A* to arrive at an optimal solution. In this thesis, we focus on the computation of potential heuristics for satisficing planning, where plan optimality is not required and the objective is to find any solution. Specifically, our focus is on the computation of potential heuristics that are descending and dead-end avoiding (DDA), since these properties guarantee favorable search behavior when used with greedy search algorithms such as hillclimbing. We formally prove that the computation of DDA heuristics is a PSPACE-complete problem and propose several approximation algorithms. Our evaluation shows that the resulting heuristics are competitive with established approaches such as Pattern Databases in terms of heuristic quality but suffer from several performance bottlenecks.

Contents

Chapter 1

Introduction

The problem of coordinating large-scale operations with many variable components is often encountered in the real world. A logistics company, for example, may be interested in transporting cargo between and inside different cities. Solving such a task usually entails searching for a sequence of actions that will lead to some desired outcome. In our example, this desired outcome could be one where all packages have been delivered to their respective destinations. The sequence of actions that leads to this outcome is called a *plan*.

In the real world, said actions may be associated with certain costs. Thus, when looking for a plan, one is typically interested in finding one with minimal total cost. Finding such optimal solutions is the main objective in the field of of *cost-optimal planning*

The problem of solving such tasks is computationally challenging because of the large number of courses of action that can be taken in an attempt to achieve the given goal. Most sequences of actions do not constitute a plan and many that do are suboptimal. In the general case, planning has been shown to be a PSPACE-complete problem (Bylander, 1994). Usually, the difficulty of a planning task grows with its size. For instance, in our logistics example, the problem of finding a plan becomes more challenging as more and more packages are added. Therefore, in many situations, it is desirable to trade off the guaranteed plan optimality against shorter solution times and lower memory costs. The variation of planning that makes this trade-off is known as *satisficing planning*. Here, the objective is to find any plan that leads from the initial situation to a goal state. It should be mentioned that, although this trade-off indeed leads to shorter solution times, the overall computational complexity of satisficing planning is the same as in the optimal case.

The most common approach to solving tasks in both, cost-optimal as well as satisficing planning, is *heuristic search*. Here, the planning task induces a graph, also known as a *transition system*, where each node represents a specific world state. In the aforementioned logistics example, a world state could be described by the present locations of all packages and vehicles. If there exists an action that can transform one world state into another then the said graph contains an edge between the two nodes.

A search algorithm is then used to find a path from the initial node to one of the nodes corresponding to a goal world state. Said search algorithms are guided by a *heuristic function* that estimates the optimal cost of reaching the goal (or, alternatively, the number of transitions needed to reach a goal state). An important result concerning the search behavior of the A* search algorithm (Hart et al., 1968) shows that a heuristic that never overestimates the optimal plan cost is guaranteed to guide A* to a cost-optimal solution. Consequently, much research in the field of cost-optimal planning is focused on developing

precise *admissible* heuristics, that is, heuristics that satisfy the aforementioned property.

In satisficing planning, on the other hand, there exists no well-established understanding of what constitutes a good heuristic function. Several possible ideas will be discussed in Sections 3.3 and 3.6.

The focus of this thesis are *potential heuristics* for satisficing planning. Potential heuristics are linear combinations of *features* of a world state. In principle, these functions can be used in both, cost-optimal and satisficing planning, assuming that appropriate feature weights are chosen. For cost-optimal planning, this usually means choosing weights such that admissibility is preserved. For the satisficing planning case, we will focus on computing heuristics that are *descending and dead-end avoiding (DDA)*.

In Chapter 3, we first give an overview of existing research on potential heuristics and related approaches. In Chapter 4, we study the general theoretical complexity of computing DDA heuristics as well as an alternative variation thereof. Our analysis shows that the computation of such heuristics is a PSPACE-complete problem. Following up on this result, we introduce various algorithms for computing potential heuristics for satisficing planning, discuss their strengths and weaknesses, and compare their performance to that of established heuristics in Chapter 6.

Chapter 2

Background

In this chapter, we provide context for the remainder of this thesis by introducing planning tasks and potential heuristics.

2.1 The SAS$^+$ Planning Formalism

In the field of classical planning, we consider tasks where the world is fully observable and all actions are deterministic, that is, the result of taking any action in any specific world state is known in advance. Specifically, we focus on tasks given in the SAS$^+$ planning formalism (Bäckström and Nebel, 1995).

A SAS$^+$ task is fully defined by the 4-tuple $\Pi = \langle V, I, O, \gamma \rangle$. Here, V is a set of all *state variables*. We will refer to state variables as *variables* whenever doing so will not cause any ambiguity. Each variable $v \in V$ is associated with a finite domain $dom(v)$. A variable-value pair $\langle v, d \rangle$ with $v \in V$ and $d \in dom(v)$ is called a *fact*. A set of facts is called a *partial assignment*. A partial assignment is called *valid* if no variable $v \in V$ is mentioned in more than one fact within said assignment. Unless stated otherwise, all partial assignments are assumed to be valid. We refer to the set of all variables mentioned in some assignment s as $vars(s)$. The value assigned to variable v by s is denoted by $s[v]$. A partial assignment s that mentions all variables (i.e. $vars(s) = V$) and thus fully describes some situation in the world, is called a *state*. The *initial state* of the world is given by the initial assignment I. We say that two partial assignments s and t *agree* if and only if the variables that they have in common are assigned to the same values, i.e. $s[v] = t[v]$ for all $v \in vars(s) \cap vars(t)$. The partial assignment γ defines the objective of the planning task by specifying values that must be achieved for certain variables. States that agree with this assignment are called *goal states*.

Set O contains all *operators* of the planning task. Operators $o \in O$ are deterministic actions that have a *precondition* $pre(o)$ and an *effect* $\mathit{eff}(o)$, where both are partial assignments. An operator o is applicable in a state s if and only if $pre(o)$ and s agree. We use $app(s)$ to refer to the set of all operators that are applicable in state s. Applying o in s leads to a *successor state* $s[\![o]\!]$ with $s[\![o]\!][v] = \mathit{eff}(o)[v]$ for all $v \in vars(\mathit{eff}(o))$ and $s[\![o]\!][v] = s[v]$ otherwise. $succ(s)$ denotes the set of all successors of s. Informally, this means that the assignment s is changed as specified by the effect of o, while variables that are not mentioned in $\mathit{eff}(o)$ retain the values they had before o was executed. Furthermore, each $o \in O$ is associated with a certain cost $cost(o) \in \mathbb{R}_0^+$.

A task is said to be in transition normal form (TNF) (Pommerening and Helmert,

2015) if $vars(\mathit{eff}(o)) = vars(\mathit{pre}(o))$ for all operators $o \in O$. We assume that all tasks are given in TNF. We can do so without loss of generality because there exists a polynomial time algorithm for converting SAS$^+$ tasks to TNF. For tasks in TNF, we use $vars(o)$ as a shorthand notation to refer to variables that are accessed by operator o.

A *path* π is a sequence of operators. We use the shorthand notation $s[\![\pi]\!]$ to denote the result of sequentially applying all $o \in \pi$ beginning in state s. State s is said to be *solvable* if there exists a path π such that $s[\![\pi]\!]$ is a goal state. Furthermore, s is *reachable* if there exists a path π with $I[\![\pi]\!] = s$. States that are both, reachable and solvable are said to be *alive*. The cost $cost(\pi)$ of a path is the sum of costs of its elements. Path π is called a *plan* if $I[\![\pi]\!]$ agrees with γ, i.e. the path leads from the initial state to a goal state.

Given a SAS$^+$ planning task, the objective of a *planner* is to find a plan. In this thesis, we differentiate between two areas of classical planning: satisficing and cost-optimal planning. In satisficing planning, a task is considered solved once any plan has been found. In cost-optimal planning, on the other hand, the planner is expected to return a plan π such that there exists no other plan τ with $cost(\tau) < cost(\pi)$, i.e. the returned plan must have minimal costs among all possible plans.

Bylander (1994) showed that planning is a PSPACE-complete problem. Specifically, he proved that the PlanEx-decision problem, that is, the problem of deciding whether there exists a solution for the given planning task, is PSPACE-complete in the general case, but may become simpler given additional assumptions about the structure of the task. PlanEx is the decision problem underlying satisficing planning. An analogous result was shown for the bounded-cost plan existence decision problem (BcPlanEx), where the question is whether or not there exists a plan whose cost does not exceed the given cost bound. This is the decision problem corresponding to cost-optimal planning.

2.2 Heuristic Search

Heuristic search is the most common approach to planning. Here, a SAS$^+$ task induces a *transition system* or *state space* that is subsequently searched for a path from the initial state to some goal state. Formally the transition system of task $\Pi = \langle V, I, O, \gamma \rangle$ is given by the 6-tuple $\mathcal{S}(\Pi) = \langle S, L, c, T, s_0, S_\star \rangle$ where

- S is the finite set of all partial assignments s with $vars(s) = V$. In other words, S is the set that holds all states of the planning task. We use the shorthand notation $|\mathcal{S}(\Pi)|$ to refer to $|S|$.
- L is a finite set of *labels*. Each label $l_o \in L$ corresponds to an operator $o \in O$. Labels associate transitions between two states with operators that induce said transitions.
- $c : L \to \mathbb{R}_0^+$ is a *label cost* function. This function corresponds to the cost function over the set O of operators, that is, $c(l_o) = cost(o)$ for all $l_o \in L$ and the corresponding $o \in O$.
- $T \subseteq S \times L \times S$ is the set of all transitions in the state space. A transition is a triple $\langle s, l_o, s' \rangle$ where $s[\![o]\!] = s'$.
- $s_0 \in S$ is the *initial state* that is equivalent to the initial assignment I of Π, and
- $S_\star$ is the set of all goal states.

Algorithms that are usually employed in planning are guided by one or multiple *heuristic functions*. A heuristic is a function $h : S \to \mathbb{R}_0 \cup \{\infty\}$ that estimates the distance of the given state to a goal state. The *perfect heuristic* h^* returns the real distance of the given state to the nearest goal state (here, path-cost and distance are synonymous).

The properties of a heuristic function have direct influence on the behavior of the search algorithm. For example, heuristics whose estimates never exceed those of h^* for any given state are called *admissible*. Admissibility is a desirable property in optimal planning because A* with an admissible heuristic is guaranteed to always find the cost-optimal path to a goal (Hart et al., 1968). The same result also holds for iterative-deepening A* (Korf, 1985).

In the satisficing planning case, there is no established understanding of what constitutes a good heuristic. In this thesis, we focus on heuristics that are *descending and dead-end avoiding (DDA)* (Seipp et al., 2016). A heuristic h is *descending* if every alive non-goal state has an alive improving successor. That is, for all non-goal alive s, there exists a $t \in succ(s)$ with $h(t) < h(s)$. Furthermore, h is called *dead-end avoiding* if $h(t) \geq h(s)$ for all non-alive $t \in succ(s)$.

Intuitively, these two properties imply that the heuristic always makes a recommendation concerning the next state to be visited and it is guaranteed that a search algorithm that follows these recommendations will never transition from a solvable state to an unsolvable one. Thus, a greedy search algorithm equipped with a DDA heuristic is guaranteed to reach a goal state without backtracking as long as the planning task is solvable. Additionally, Seipp et al. (2016) show that, given a DDA heuristic h, a solution will be found after at most $\mathcal{L} = h(s_0) - \min_{s \in S} h(s)$ search node expansions.

2.3 Potential Heuristics

The focus of this thesis is on potential heuristics (Pommerening et al., 2015). Here, a potential heuristic is a weighted sum of indicator functions, where each indicator function tests whether a specific *conjunctive feature* is present in the given state. Seipp et al. (2016) formally define the potential heuristic h^{pot} as

$$h^{\text{pot}}(s) = \sum_{F \in \mathcal{F}} w(F)[F \subseteq s] \tag{2.1}$$

Here, $[\cdot]$ denotes the indicator function (Knuth, 1992) and $w(F)$ is the numerical weight (the *potential*) of *feature* F, which is a *set of facts*. $\mathcal{F}$ denotes the set of all features.

The cardinality $|F|$ of a feature is referred to as its *dimension*. Features with a dimension of 1 are called *atomic features*. Seipp et al. (2016) define the *dimension of a potential heuristic* as the highest dimension of the features considered by the heuristic. The notion of dimensionality is directly related to evaluation time: a potential heuristic of dimension d for the planning task $\langle V, I, O, \gamma \rangle$ can be evaluated in $O(|V|^d)$ time. A review of existing research on potential heuristics is given in the next chapter.

Chapter 3

Related Work

Since their introduction by Pommerening et al. (2015), potential heuristics have been successfully used in classical planning. Typically, one of the crucial first steps towards successfully employing a potential heuristic is the computation of feature weights $w(F)$ for all $F \in \mathcal{F}$. The other important problem is that of automatically finding a good set of features $\mathcal{F}$. Much of the existing research on potential heuristics focuses on the former task. In this chapter, we provide an overview of these approaches for both, optimal as well as satisficing planning.

3.1 One-Dimensional Potential Heuristics

Potential heuristics provide a general framework that can be used to represent a number of different well-known heuristics. For example, a simple goal-count heuristic (Fikes and Nilsson, 1971) can be represented using a set of atomic features $\mathcal{F} := \{\langle v, d\rangle \mid v \in \mathit{vars}(\gamma), d \neq \gamma[v]\}$ by setting $w(F) := 1$ for all $F \in \mathcal{F}$.

Ferenczi (2016) uses potential heuristics to approximate Pattern Database (PDB) heuristics (Culberson and Schaeffer, 1998). To this end, he formulates the problem of finding potentials for the chosen set $\mathcal{F}$ of atomic features as a linear regression problem. A problem of this approach is that the estimates obtained in this way, unlike PDBs, provide no admissibility guarantees, which in turn means that A* is not guaranteed to find an optimal plan given such a heuristic.

Instead of basing potential computations on some prototype heuristic, Pommerening et al. (2015) opt to directly find potentials for atomic features such that the resulting heuristic is always admissible. To this end, they demand that the heuristic be always *goal-aware* and *consistent*.

A heuristic h is goal-aware if $h(s_\star) = 0$ for all goal states $s_\star$. It is consistent if $h(s) \leq \mathit{cost}(o) + h(s[\![o]\!])$ for every state and the corresponding applicable operators. Together, consistency and goal-awareness imply admissibility (Russell and Norvig, 1995).

The problem of finding potentials such that the resulting potential heuristic has the aforementioned two properties is then formulated as a linear programming (LP) problem. The constraints follow from the two desired properties. We write the constraint for goal-awareness as $\sum_{F \in \mathcal{F}} w(F)[F \subseteq s_\star] \leq 0$. The constraint for consistency is obtained by rewriting the definition of consistency: $h^{\mathrm{pot}}(s) - h^{\mathrm{pot}}(s[\![o]\!]) \leq \mathit{cost}(o)$ for all s and all $o \in \mathit{app}(s)$. Pommerening et al. (2015, 2017) observe that, given a fixed operator o, the set of atomic features $\mathcal{F}$ can be divided into *irrelevant features* $\mathcal{F}^{irr}$ whose variables are not

changed by o (i.e. $vars(F) \not\subseteq vars(o)$ for all $F \in \mathcal{F}^{irr}$) and *context-independent features* $\mathcal{F}^{ind}$ with $vars(F) \subseteq vars(o)$ for all $F \in \mathcal{F}^{ind}$. For atomic features, $|vars(F)| = 1$ and thus these are the only possible cases.

Given this observation, they conclude that the consistency constraint for atomic features depends only on $o \in O$, but not on states s. This is due to the fact that the contributions of irrelevant features to $h^{\text{pot}}(s)$ and $h^{\text{pot}}(s[\![o]\!])$ cancel each other out, while the contributions of context-independent features depend solely on $pre(o)$ and $\mathit{eff}(o)$. Thus, the final LP constraints are:

$$cost(o) \geq \sum_{v \in vars(o)} \Big(w(\{\langle v, pre(o)[v] \rangle\}) - w(\{\langle v, \mathit{eff}(o)[v] \rangle\}) \Big) \tag{3.1}$$

$$0 \geq \sum_{v \in V} w(\{\langle v, s_\star[v] \rangle\}) \tag{3.2}$$

This formulation does not yet specify the objective opt that is to be optimized. To address this, Pommerening et al. (2015) introduce the objective function opt_{s_I} that represents the heuristic estimate of the initial state:

$$\text{opt}_{s_I} = \sum_{v \in V} w(\{\langle v, s_I[v] \rangle\}) \tag{3.3}$$

Furthermore, they show that the potential heuristic $h^{\text{pot}}_{\text{opt}_{s_I}}$ obtained by solving the resulting linear program has the same heuristic estimate for the initial state as the State Equation heuristic h^{SEQ} (Bonet, 2013).

A weakness of the above LP formulation is that it is only concerned with computing the highest admissible heuristic estimate for the initial state. For this reason, a potential heuristic computed with this objective will likely provide less informative estimates for other encountered states. To alleviate this issue, Seipp et al. (2015) propose several alternative objective functions.

The first objective they introduce ($\text{opt}_{\mathcal{S}}$) aims to maximize the average heuristic value across the entire set S of all states of the planning task. The challenge, here, is that dead-end states (i.e. states s with $h^*(s) = \infty$) can cause the linear program to become unbounded. To prevent this, they introduce an upper bound M on all potentials by adding constraints of the form $\text{pot}(\langle v, d \rangle) \leq M$ for all $\langle v, d \rangle \in \mathcal{F}$, thus ensuring that the LP remains bounded.

Another challenge is that heuristic values of unreachable parts of the state space (i.e. states that will never be generated by A* search) influence the solution. To avoid this, Seipp et al. (2015) propose optimizing the average heuristic of *reachable states*. Since there is no known efficient way of extracting only the reachable portion of the transition system, their proposed objective function $\text{opt}_{\hat{S}}$ aims to maximize the average over some set $\hat{S}$ of sample states that are known to be reachable. Lastly, they also introduce an objective function for minimizing the search effort using a formula by Korf et al. (2001).

Experimental evaluation shows that focusing only on the reachable parts of the state space leads to more tasks being solved, when compared to optimizing for other objectives. Even better performance, however, is achieved by using the maximum of multiple potential heuristics as the final estimate.

3.2 Higher-Dimensional Potential Heuristics

The analysis of several planning domains by Seipp et al. (2016) (see Section 3.3) shows that potential heuristics with only atomic features often fail to account for interactions between different facts of the planning task. Thus, higher-dimensional features may be necessary in order to obtain a more informative potential heuristic.

Pommerening et al. (2017) generalize the LP constraints introduced in the previous section to also account for features that are conjunctions of two facts. While, as in the one-dimensional case, the set of features $\mathcal{F}$ can be partitioned into features $\mathcal{F}^{irr}$ that are irrelevant for some fixed $o \in O$ as well as context-independent features $\mathcal{F}^{ind}$, the higher-dimensional LP formulation must also account for a third type of features, namely *context-dependent features*.

A conjunctive 2-dimensional feature F is considered context-dependent (formally, $F \in \mathcal{F}^{ctx}$) if it consists of two facts f_o and $f_{\bar{o}}$, where $vars(f_o) \in vars(o)$ and $vars(f_{\bar{o}}) \notin vars(o)$. Intuitively, this means that F consists of a context-independent feature f_o and an irrelevant feature $f_{\bar{o}}$. Due to this irrelevant feature, it is not possible to tell whether F is present in s or not without evaluating $[f_{\bar{o}} \in s]$. Thus consistency constraints are no longer state-independent.

Pommerening et al. (2017) show that, in the case of 2-dimensional heuristics, accounting for context-dependent features in the consistency condition leads to an LP that can still be represented by a compact set of linear constraints. Specifically, they show that the total number of constraints in such a problem is $O(|O||V|d)$ where d is the upper bound on the variable domain sizes.

Furthermore, they prove that a compact representation is not possible for conjunctive features of dimension 3 or more and that testing whether a corresponding potential heuristic is consistent is a coNP-complete problem. However, they also introduce a fixed-parameter tractable algorithm with which it is still feasible to compute the necessary constraints in certain special cases.

3.3 Heuristics in Satisficing Planning

In satisficing planning, plan optimality is not a requirement and the goal is to find any valid solution for the planning task. Therefore, demanding that heuristics be admissible offers no advantages and only unnecessarily restricts heuristic quality. For this reason, the LP model introduced earlier is is not appropriate for satisficing planning.

Instead of pursuing admissibility, Seipp et al. (2016) introduce the concept of DDA heuristics (see Section 2.2) and study the conditions under which there can exist a DDA potential heuristic. Specifically, they introduce *correlation complexity* as a measure of how complex a potential heuristic must be in order to satisfy these favorable properties in the given planning task. Correlation complexity is defined as the minimum dimension d of all DDA potential heuristics for planning task Π. This concept is further extended to characterize entire planning domains: the correlation complexity of a planning domain $\mathcal{D}$ is the maximal correlation complexity among all planning tasks $\Pi \in \mathcal{D}$. Unlike the correlation complexity of a single task, which is bounded by the number n of variables in said task, the complexity of a domain can be infinite.

Given a polynomial p and a task with encoding size n from a domain with a finite correlation complexity, Seipp et al. (2016) show that the time needed to find a plan is bounded by a polynomial in n under the following two conditions: a descending and dead-end avoiding potential heuristic can be computed in $p(n)$ time and the corresponding

integer-valued feature weights are polynomially bounded: $|w(F)| \leq p(n)$.

Although they introduce said tractability result and shows how correlation complexity can be used to prove the existence of descending and dead-end avoiding potential heuristics of a specific complexity, the question of how to automatically find such heuristics remains open. This is also one of the central questions of this thesis.

3.4 Generalized Potential Heuristics

A possible approach to the task of automatically finding good potential heuristics for satisficing planning was developed by Francès et al. (2019). Ideas from their work form the basis for all algorithms introduced in this thesis. They focus on finding *generalized potential heuristics*, that is, heuristics that work for any instance of a planning domain. To compute such a heuristic, they use small instances from the chosen domain to learn weights for some candidate set of features $\mathcal{F}$.

Let $\mathcal{S}(\Pi) = \langle S, L, c, T, s_0, S_\star \rangle$ be the transition system of the given task Π. The set S_A is defined as the subset of S that contains only alive states. Furthermore, $T_D \subset T$ is the subset of transitions that lead to an unsolvable state. Given a complexity measure[1] $\mathcal{K}(F)$ defined for all features $F \in \mathcal{F}$, they formulate the problem of finding a descending and dead-end avoiding heuristic as a mixed-integer programming (MIP) problem:

$$\min \sum_{F \in \mathcal{F}} [w(F) \neq 0] \mathcal{K}(F) \tag{3.4}$$

$$\text{s.t.} \bigvee_{s' \in succ(s)} h(s') + 1 \leq h(s) \quad \text{for } s \in S_A \tag{3.5}$$

$$h(s') \geq h(s) \quad \text{for } \langle s, l, s' \rangle \in T_D \tag{3.6}$$

Intuitively, this MIP aims to minimize the total complexity of all features $F \in \mathcal{F}$ with non-zero potentials, that is, all features that contribute to the heuristic estimate. Constraint (3.5) demands that the heuristic be descending, while (3.6) constraints the MIP to only admit dead-end avoiding heuristics as solutions.

The disjunction in (3.5) is encoded in modern MIP solvers such as IBM CPLEX using *indicator constraints*. That is, a disjunction $\bigvee_i^n \varphi_i$ over n linear constraints φ_i is replaced with a set of constraints $l_i \rightarrow \varphi_i$, where l_i is a binary *indicator variable*. Constraint φ_i is only active if the corresponding indicator variable equals to 1. The disjunction is then achieved by adding the linear constraint $\sum_i^n l_i \geq 1$, which demands that at least one constraint φ_i be satisfied.

The obvious reasons, why using this MIP for anything but the smallest task instances is difficult, are the need to compute the sets S_A and T_D that induce all constraints of the model as well as the inherent NP-hardness of mixed-integer programming.

Instead of computing MIP constraints for the entire set S right away, Francès et al. (2019) opt to generate them incrementally within a *constraint generation loop*. In this loop, an initial subset S^0 of S is randomly selected. The solution of the corresponding MIP is then tested for descendingness and dead-end avoidance on the remaining states in S. If there are states where these properties do not hold, then set S^0 is extended with χ such states and the MIP is solved again, but this time with constraints induced by this updated set.

[1] See Francès et al. (2019) for a detailed description of this measure.

In the worst case, this approach will converge to the initial case, where constraints are computed from the entire S. However, Francès et al. (2019) observe that, in many cases, a generalized heuristic can be found before that. If the MIP turns out to be unsolvable, then a generalized heuristic with the desired properties does not exist for the given feature set $\mathcal{F}$. In this case, one can start over with more complex features.

3.5 Perfect Potential Heuristics

Analogously to the complexity analysis for satisficing planning by Seipp et al. (2016), Corrêa and Pommerening (2019) perform an empirical study of the complexity that a potential heuristic must have in order to accurately represent the perfect heuristic h^*.

Potential heuristics, as introduced by Pommerening et al. (2015), are too restrictive to represent h^* for all tasks, because they are finite by definition, and thus cannot cover cases where $h^*(s) = \infty$. Therefore, Corrêa and Pommerening (2019) define the *perfect potential heuristic* as the combination of two potential heuristics.

$$h^{\text{pot}}_{w_1,w_2}(s) = \begin{cases} \infty & \text{if } h^{\text{pot}}_{w_2}(s) > 0 \\ h^{\text{pot}}_{w_1}(s) & \text{otherwise.} \end{cases} \tag{3.7}$$

where $h^{\text{pot}}_{w_1}(s) = h^*(s)$ for all solvable states and $h^{\text{pot}}_{w_2}(s) > 0$ if and only if s is unsolvable. Building on the idea of correlation complexity, they introduce *optimal correlation complexity* as the minimal dimension of a perfect potential heuristic for task Π.

In their study, they compute the optimal correlation complexities of small tasks by first computing the corresponding perfect heuristics. Afterwards they use an iterative approach that generates a set $\mathcal{F}_n$ of all features up to dimension n for $1 \leq n \leq |V|$. For each generated set of features, they solve an LP with constraints of the form $\sum_{F \in \mathcal{F}_n} w(f)[F \subseteq s] = h^*(s)$ for all solvable $s \in \mathcal{S}$, thus ensuring that the potential heuristic is indeed perfect on the solvable portion of the state space. An analogous approach is used to compute the heuristic for detecting unsolvable states. The first value of n for which the LP has a feasible solution is the optimal correlation complexity of the planning task.

Their experiments show a number of interesting results. First of all, even domains such as GRIPPER and VISITALL, that are considered easy[2] have high lower bounds on optimal correlation complexity. The authors observe that this problem is partially caused by states that are solvable, yet unreachable, because they violate mutexes. For such states, higher dimensional features may be needed to capture their heuristics. By excluding such states from consideration and focusing only on the reachable part of the state space they could reduce the optimal correlation complexity of a number of tasks as well as greatly decrease the total number of features needed to replicate the perfect heuristic.

In addition to the aforementioned LP approach, Corrêa and Pommerening (2019) introduce an iterative algorithm that minimizes the total absolute error $E(h) = \sum_{s \in \mathcal{S}} |h^*(s) - h(s)|$ between the perfect heuristic and the current potential heuristic h by extending the latter with new features and greedily selecting weights for them. Experimental evaluation shows that the highest decrease in the total absolute error comes from the addition of a small number of high-dimensional features. This has great practical implications, as it means that users with domain knowledge can construct informative heuristics by manually selecting a small number of said high-dimensional features and then computing weights for them using the fixed-parameter tractable algorithm from Pommerening et al. (2017).

[2]Seipp et al. (2016) prove that these domains have a correlation complexity of 2.

3.6 Goal Distance Rank Correlation

While Seipp et al. (2016) showed that descending and dead-end avoiding heuristics are potentially beneficial for greedy search, Wilt and Ruml (2015) discovered a quantitative metric of heuristic quality for satisficing planning. They observed that, in satisficing planning, the exact values returned by the heuristic are of little importance. What is truly important is the ordering of states induced by said heuristic values. Generally, a heuristic can return an arbitrary heuristic value for state s as long as any state s' that is closer to the goal has a lower heuristic value. In essence, they propose measuring the quality of a heuristic in terms of its ability to replicate the state-ranking induced by the perfect heuristic.

Specifically, Wilt and Ruml (2015) propose using Kendall's τ coefficient (Kendall, 1948) to quantify the degree to which the given heuristic replicates the ranking induced by the perfect heuristic. They call this measure the *Goal Distance Rank Correlation (GDRC)*.

A GDRC value of 1 implies that the state ranking induced by the heuristic in question exactly matches that of the perfect heuristic for task Π. In the opposite case, where the ranking of any two states differs from that of the perfect heuristic, the GDRC value is -1. Because the perfect heuristic is DDA (assuming that there are no zero-cost operators), a heuristic with a GDRC score of 1 is also DDA. In fact, such a heuristic is DDA on the entire family of tasks obtained by replacing the initial state of Π with any other state of its state space.

On the other hand, however, a heuristic that is DDA does not necessarily have a positive GDRC value. The DDA set of properties pertains only to the reachable portion of the transition system. A heuristic that always ranks correctly on this reachable portion is DDA, even though it might rank incorrectly in the unreachable parts. In other words, a GDRC score of 1 is a sufficient, but not a necessary condition for a DDA heuristic.

3.7 GLEM

Outside of the field of classical planning, Buro (1998) developed the *Generalized Linear Evaluation Model (GLEM)*, a framework for representing position evaluation functions for games as linear combinations of features. Formula 3.8 illustrates this framework in our notation.

$$e(s) = g(\sum_{F \in \mathcal{F}} w(F)[F \subseteq s]) \tag{3.8}$$

It is easy to see that a potential heuristic can be represented in GLEM by using the identity function $g(x) = x$. In his work, Buro (1998) addresses several practical considerations concerning the computation of such evaluation functions using machine learning. Most notably he proposes using a sample of states not only for learning feature weights, but also for the process of feature selection itself. As will be discussed in Section 5.1, our implementation of the feature generation process builds on this idea.

Chapter 4

Theoretical Results

A well known result in planning is one by Bylander (1994) that shows that classical planning is a PSPACE-complete problem. A more recent result by Seipp et al. (2016) (see Section 3.3) proves that given a DDA heuristic a satisficing solution will be found after a polynomial number of state expansions. The necessary condition for this is, that $h(I) - \min_{s \in S} h(s)$ is bounded by $p(n)$, where p is some polynomial and n is the encoding size of the planning task. Clearly, this result makes DDA heuristics attractive for satisficing planning. However, it also begs the question of how hard it is to obtain such heuristics. Therefore, in this chapter, we analyse the complexity of computing a DDA heuristic and also consider heuristics with an alternative set of properties. Our analysis shows that the problem of ensuring these properties is generally not tractable.

4.1 DDA Heuristics

As already informally discussed in Section 2.2, we consider a heuristic to be DDA if it satisfies the following two properties:

$$\forall s \in (S_A \setminus S_\star) : (\exists s' \in succ(s) : h(s') < h(s)) \tag{4.1}$$

$$\forall s \in S_A : (\forall s' \in (succ(s) \setminus S_A) : h(s') \geq h(s)) \tag{4.2}$$

where S_A is the set of all alive states of the given task Π. The first formula demands that the heuristic be descending, that is, each alive state must have a successor with a better heuristic value. The second formula demands that this improving successor is never an unsolvable state. If a task is unsolvable then $S_A = \emptyset$ as no state is both reachable and solvable. Here, the above formulas are true for all h and we say that the heuristic is *trivially DDA*.

Furthermore, we formally define the IsDDA decision problem as follows:

GIVEN: task $\Pi = \langle V, I, O, \gamma \rangle$ and a heuristic h
QUESTION: is h descending and dead-end avoiding (DDA) in Π?

Theorem 1. *The* IsDDA *decision problem is PSPACE-complete.*

Proof. Since PSPACE=coPSPACE, this result follows from the proof that the complement of IsDDA (i.e. the problem of deciding whether a heuristic is *not* DDA) is PSPACE-complete (see Theorems 2 and 3). □

Theorem 2. *The* NotDDA *decision problem (complement of* IsDDA*) is PSPACE-hard.*

Proof. We show PSPACE-hardness by reduction from the plan existence problem PlanEx (Bylander, 1994): Given a task Π for which we want to decide PlanEx, construct a heuristic function $\hat{h}$ that cannot be DDA in solvable tasks (e.g. $\hat{h}(s) = 0 \quad \forall s \in S$). If $\Pi \notin$ PlanEx then this means that Π has no alive states, which implies that $\hat{h}$ is trivially DDA and therefore $\langle \Pi, \hat{h} \rangle \notin$ NotDDA. Inversely, if $\Pi \in$ PlanEx, then there exist alive states and $\hat{h}$ will be inevitably found to not satisfy conditions 4.1 or 4.2, which means that $\langle \Pi, \hat{h} \rangle \in$ NotDDA. □

The proof of Theorem 2 does not address the edge case where the planning task is initially solved, that is, the initial state is also a goal state. We can ignore such cases without loss of generality because such trivial tasks can be efficiently detected and transformed into equivalent non-trivial tasks.

Theorem 3. *The* NotDDA *decision problem is PSPACE-complete.*

Proof. With hardness proven in Theorem 2, only PSPACE membership remains to be shown. A possible approach to deciding NotDDA in polynomial space is illustrated in Algorithm 1. Here, the algorithm systematically generates all states in the state space. Whenever a new state is generated, the previous state is discarded, thus ensuring that the memory consumption remains polynomial in V even though the time complexity is exponential. For each generated alive state, the algorithm iterates over all of its successors and verifies that the given heuristic satisfies the DDA properties. □

Algorithm 1: Deciding NotDDA in polynomial space.

Given : task $\Pi = \langle V, I, O, \gamma \rangle$ and heuristic h
Question: is h *not* DDA?

for $i \leftarrow 1 \,..\, |\mathcal{S}(\Pi)|$ **do**
  if s_i *is not alive* **then**
    continue;
  end
  has_improving_succ $\leftarrow$ **F**;
  for $o \in app(s_i)$ **do**
    if $[s_i[\![o]\!]$ *is alive*$] \wedge [h(s_i[\![o]\!]) < h(s_i)]$ **then**
      has_improving_succ $\leftarrow$ **T**;
    end
    if $[s_i[\![o]\!]$ *is not alive*$] \wedge [h(s_i[\![o]\!]) < h(s_i)]$ **then**
      accept; // h is not dead-end avoiding ⇒ not DDA.
    end
  end
  if ¬*has_improving_succ* **then**
    accept; // h is not descending ⇒ also not DDA.
  end
end
fail; // All DDA conditions satisfied. h is DDA.

Thus, Theorem 1 shows that the problem of finding a DDA heuristic for the given planning task is as hard as the problem of finding a plan for said task. To identify parts of the DDA problem that are most likely responsible for this hardness result we reformulate IsDDA as a Quantified Boolean Formula (QBF):

$$\forall s \Big([s \textit{ is goal}] \vee \Big([s \textit{ is alive}] \rightarrow \big(d(s) \wedge da(s) \big) \Big) \Big) \tag{4.3}$$

where $[\cdot]$ is a predicate, s is a short-hand notation for all state variables in V (e.g. $\forall s$ is equivalent to $\forall v_1 v_2 ...$ if $V = \{v_1, v_2, ...\}$), and $d(s)$ (resp. $da(s)$) encodes descendingness (resp. dead-end avoidance):

$$d(s) = \exists s' \Big([s' \in succ(s)] \wedge [h(s') < h(s)] \Big) \tag{4.4}$$

$$da(s) = \neg \exists s' \Big([s' \in succ(s)] \wedge \neg [s' \textit{ is solvable}] \wedge [h(s') < h(s)] \Big) \tag{4.5}$$

A closer look at d and da shows that the predicates $[s \textit{ is alive}]$ and $\neg[s' \textit{ is solvable}]$ are PSPACE-hard to evaluate. The latter is obviously the complement of the PlanEx decision problem, while the former has a simple reduction from PlanEx: note that $\langle V, I, O, \gamma \rangle \in$ PlanEx if and only if I is alive. Furthermore, as established earlier, $\langle V, I, O, \gamma \rangle \notin$ PlanEx means that there are no alive states and thus I cannot be alive. Whether a state s of some task $\langle V, I, O, \gamma \rangle$ is alive or not can be decided in polynomial space by deciding PlanEx for tasks $\langle V, I, O, s \rangle$ and $\langle V, s, O, \gamma \rangle$.

4.2 DER Heuristics

One of the advantages of representing decision problems as QBFs is the framework they provide for reasoning about the complexity of the underlying decision problem. Specifically, it has been shown (Stockmeyer, 1976; Wrathall, 1976) that the complexity of deciding satisfiability of a QBF is directly linked to the *prefix type* (Kleine Büning and Bubeck, 2009) of the formula.

A QBF where $\forall$ (respectively $\exists$) is the outer-most quantifier, has the prefix type Π_n (respectively Σ_n) where n denotes the number of alternations between the two quantifier-types. The prefix type is easiest to determine when the QBF is given in *prenex form*. This form is characterized by all quantifiers appearing at the beginning of the formula. These quantifiers are then followed by a quantifier-free formula called a *matrix*. For example, the formula $\exists a \forall b \exists c \Phi$ with matrix Φ has the prefix type Σ_3. Every propositional formula is of the prefix type $\Sigma_0 = \Pi_0$.

The various prefix types correspond to different complexity classes on the *polynomial-time hierarchy* (Meyer and Stockmeyer, 1972) that are defined as follows:

$$\Sigma^P_{k+1} := NP^{\Sigma^P_k}, \quad \Pi^P_{k+1} := co\Sigma^P_{k+1}$$

Σ^P_{k+1} is the class of all problems that can be decided non-deterministically in polynomial time assuming the availability of an oracle that can decide problems in Σ^P_k in $O(1)$ time. Π^P_{k+1} is the class that contains all problems that are complements of those from Σ^P_{k+1}. Most notably Σ^P_1 corresponds to the complexity class NP, and $\Pi^P_1 =$ coNP.

The results from Section 4.1 show that computing exact DDA heuristics is prohibitively expensive. This calls for a search of alternative properties with an easier underlying decision problem, that is, a decision problem that lies in one of the complexity classes of

the polynomial-time hierarchy. Therefore, we investigated the use of *dead-end recognizing (DER)* heuristics. A heuristic h^{DER} is dead-end recognizing iff $h^{\mathrm{DER}}(s) > 0$ for all dead-end states s. Thus, a dead-end recognizing heuristic is one that correctly recognizes all dead-end states, but may also mark solvable states as dead-ends.

Assuming the existence of such a heuristic, Formula 4.3 simplifies to

$$\forall s \Big([s \textit{ is goal}] \vee \Big([h^{\mathrm{DER}}(s) = 0] \rightarrow \hat{d}(s) \Big) \Big) \tag{4.6}$$

$$\hat{d}(s) = \exists s' \Big([s' \in \mathit{succ}(s)] \wedge [h^{\mathrm{DER}}(s') = 0] \wedge [h(s') < h(s)] \Big) \tag{4.7}$$

Thus, given a fixed DER heuristic, the satisfiability of Formula 4.6 can be decided in $\mathrm{coNP}^{\mathrm{NP}}$. Unfortunately, however, the overall problem does not become easier if a DER heuristic is not given and needs to be synthesized first. Let IsDER be a decision problem that is defined as follows:

GIVEN: task $\Pi = \langle V, I, O, \gamma \rangle$ and a heuristic h
QUESTION: is h dead-end recognizing (DER) in Π?

Theorem 4. *The* IsDER *decision problem is PSPACE-hard.*

Proof. Reduction from PlanEx: We construct the heuristic

$$h(s) = \begin{cases} 0 & \text{if } s = I \\ 1 & \text{otherwise} \end{cases}$$

If $\Pi \in$ PlanEx then I is certainly solvable and h does not violate the DER property by marking it as such, i.e. $\langle h, \Pi \rangle \in$ IsDER. If $\Pi \notin$ PlanEx then I cannot be solvable, which means that h failed to recognize a dead-end and therefore $\langle h, \Pi \rangle \notin$ IsDER. □

Theorem 5. *The* IsDER *decision problem is PSPACE-complete.*

Proof. Analogously to Algorithm 1, Algorithm 2 shows how IsDER can be decided in polynomial space. PSPACE-completeness follows from this result and Theorem 4. □

Algorithm 2: Deciding IsDER in polynomial space.

```
Given: task Π = ⟨V, I, O, γ⟩ and heuristic h
Query: is h DER?
for i ← 1 .. |S(Π)| do
    if [⟨V, s_i, O, γ⟩ ∉ PlanEx] ∧ [h(s_i) ≤ 0] then
        fail;
    end
end
accept;
```

Chapter 5

Computing Potential Heuristics

In the previous chapter, we have shown that the exact computation of DDA heuristics is a PSPACE-complete problem and, thus, as hard as planning itself. Therefore, the focus of this chapter is on practical approaches for computing potential heuristics to be used in satisficing planning. The general framework shared by most of said approaches is illustrated in Algorithm 3. The concrete algorithms differ in their implementations of the BuildOptimizationProblem component, while our approach to feature generation (see Section 5.1) remains largely the same across all algorithms. Beginning in Section 5.2, we present and discuss all approaches we implemented.

Algorithm 3: Framework for computing potential heuristics.

Input : task $\Pi = \langle V, I, O, \gamma \rangle$
Output: potential heuristic h^{pot}

```
1 F ← GenerateFeatures(Π);
2 P ← BuildOptimizationProblem(Π, F);
3 h^pot ← Solve(P);
4 return h^pot;
```

5.1 Generating Features

When computing potential heuristics, feature selection is the first major challenge. The most straight-forward approach, that is also used in most of the surveyed literature, is to systematically generate all features of specific dimensions. The biggest problem with this approach is its inability to scale to larger tasks or higher feature dimensions. Furthermore, many of the features generated using this approach may never be encountered during progression search because they contain mutually exclusive facts. Such features only increase the complexity of the optimization problem.

The opposite approach is to select all features manually. Doing so effectively, requires the user to have a good understanding of the given planning domain and involves a significant amount of work in instances with many variables. Furthermore, this would turn heuristic computation into a domain-dependent approach.

As for the automatic generation part, we build on ideas from Buro (1998). Specifically, we begin by systematically generating features of the specified dimensionality, and then

use a sample of 1000 states from the state space of the task to count the number of times each generated feature was encountered in the sampled states.

Features that were not observed sufficiently often are subsequently removed from the feature space. Depending on the appearance threshold used to filter out features, the size of the feature set can be greatly reduced. In the baseline implementation, this threshold is set to 0, thus admitting all systematically generated features into the final feature set.

The sample for this procedure is generated as follows: We use the FF heuristic (Hoffmann and Nebel, 2001) to obtain a solution cost estimate $\hat{h}$ of the planning task. Subsequently, random walks are performed either in forward or in backward direction (we discuss backward sampling in Section 5.3), that is, either starting in the initial state, or starting in some random goal state. Upon each random walk, the last visited state is added to the sample. The walk direction is selected uniformly at random. This is done in order to ensure that the sample contains states that are near the initial state, as well as those that are close to some goal. The distance to be walked is also selected uniformly from the range $[0, \hat{h}]$.

5.2 Exact Potential DDA Heuristics

In Chapter 4, we showed that the pursuit of heuristics that are descending and dead-end avoiding is intractable. Nonetheless, these heuristics are desirable because they can guarantee an upper bound on the number of expansions the search algorithm will need to find a goal. It is currently unclear how DDA heuristics influence other aspects of greedy search behavior, such as the quality of the discovered solutions. In order to study this further, we describe a naive algorithm for computing potential DDA heuristics.

The algorithm builds on the idea of formulating a MIP problem similar to the one used by Francès et al. (2019) (see Section 3.4). Because we aim to compute DDA heuristics for individual tasks rather than entire domains, unlike Francès et al. (2019), our features are conjunctions of facts that may appear in some states of a planning task.

Thus, we cannot re-use the capacity measure $\mathcal{K}$ of Francès et al. (2019) for the objective function. Ideally, we would prefer to minimize the upper bound $\mathcal{L} = h(I) - \min_{s \in S} h(s)$ on the number of expansions. This objective, however, is clearly non-linear. Instead, we propose two alternative formulations. The simplest idea is to make the objective function constant:

$$\min \quad 0 \tag{5.1}$$

$$\text{s.t.} \quad \bigvee_{s' \in succ(s)} h(s') + 1 \leq h(s) \quad \text{for } s \in S_A \tag{5.2}$$

$$h(s') \geq h(s) \quad \text{for } \langle s, s' \rangle \in T_D \tag{5.3}$$

With this formulation, the MIP solver will stop upon finding the first solution that satisfies the constraints. Experiments with this approach show that the solver often fails to find an initial feasible solution under the given memory constraints. It is understandable that finding this initial solution would be challenging, since any feasible assignment of weights would also be the optimal solution under the given objective function. To address this issue, we introduce slack variables to our model:

$$\min \quad \sum_{s \in S_A} \alpha_s + \sum_{\langle s,s' \rangle \in T_D} \beta_{(s,s')} \tag{5.4}$$

$$\text{s.t.} \quad \bigvee_{s' \in succ(s)} h(s') + 1 - \alpha_s \leq h(s) \quad \text{for } s \in S_A \tag{5.5}$$

$$h(s') + \beta_{\langle s,s' \rangle} \geq h(s) \quad \text{for } \langle s, s' \rangle \in T_D \tag{5.6}$$

$$\alpha_s \geq 0 \quad \text{for } s \in S_A \tag{5.7}$$

$$\beta_{\langle s,s' \rangle} \geq 0 \quad \text{for } \langle s, s' \rangle \in T_D \tag{5.8}$$

With this formulation, an obvious feasible solution is one where the slack variables are so large that all constraints become satisfied independently from the weights assigned to the features. As the solver finds variable assignments that satisfy more of the constraints, an increasing number of slack variables can be set to zero.

Informal experiments have shown that the MIP solver is able to get reasonably close to finding an optimal solution in about half of the time allocated for optimization. Specifically, this means that the solver is able to eliminate most slack variables from the model. The other half of the time is then spent trying to eliminate the remaining few constraints (that is, constraints that are presently only satisfied by using slack variables). This means that the solver is able to compute a good/near-perfect approximation of a DDA heuristic much quicker than it is able to find the actual DDA heuristic.

With this realization, the obvious first approach for approximating DDA heuristics is to limit the time for solving the model and then work with the weights that could be computed in that time.

Another advantage of this approach is that it works even if the dimensionality of the desired potential heuristic is lower than the correlation complexity of the planning task. In such cases, it is still possible to obtain an approximation. The approach without slack variables, on the other hand would fail if the feature dimensions are too low, as this would make the problem infeasible.

Expectedly, however, the main challenge of this approach lies not in solving the MIP, but rather in formulating and storing it in memory. Indeed, in the overwhelming majority of cases, heuristic computation fails before finishing the BuildOptimizationProblem procedure. Clearly, both, the number of alive states, as well as the number of transitions to unsolvable states may be exponential in the number of state variables. This implies that the size of the MIP model itself is exponential in the number of variables. Furthermore, as mentioned earlier, deciding if a state is alive is a PSPACE-complete problem. Thus this approach is limited only to the smallest tasks.

5.3 Approximating DDA Heuristics using a Sample

As mentioned earlier, the exponential growth of the alive portion of the transition system makes the exact approach infeasible. A possible alternative is to build an optimization problem that is based only a subset of states. In the following sections, we describe two algorithms that build a fixed size sample of the state space using either forward random walks from the initial state or backward random walks from random goal states and then use the sample to approximate a DDA heuristic.

Forward-Sampling Approach

When sampling states by performing a random walk that originates in the initial state, we are guaranteed to only sample states that are reachable. Thus, with this approach, one has to assume that the other property required for a state to be alive is given. That is, states sampled by walking from the initial state are assumed to be solvable.

This assumption is incorrect in general, but holds in cases where the task is solvable and either undirected or harmless (Hoffmann, 2002). Solvable tasks that are undirected or harmless have no reachable dead-ends, thus making every reachable state solvable. A number of well known planning domains, such as BLOCKSWORLD, the n-puzzle and Rubik's cube fall into the above category.

Similarly to the sampling approach for feature selection, the sampling process begins by generating an FF estimate $\hat{h}$ for the initial state. This estimate serves as the upper bound on the distance of the random walk. The actual distance to be walked is then drawn uniformly from the interval $[0, \hat{h}]$. This is done in order to ensure that sampled states are located at various distances away from the initial state. Upon each walk, the last visited state is added to the sample set S_{sample}. The walks are repeated until the desired sample size is reached.

With the sample in place, the algorithm proceeds to construct the MIP model from Section 5.2 where the set S_A of alive states is substituted with S_{sample}. Furthermore, dead-end avoidance constraints are omitted because we do not know which successors of the sampled states are unsolvable.

This approach already avoids one of the major problems with the exact algorithm, namely the explosive state space growth. Another major bottleneck, however, stems from the NP-hardness of mixed-integer programming and remains unaddressed by this approach.

Backward-Sampling Approach

Obtaining a sample by walking backwards from some goal has proven to be a lot more problematic than the forward-sampling approach from the previous section. Generally, the assumption that a random goal and its predecessors are reachable seems to be wrong more often than not. This is due to the fact that, in most planning tasks, the goal is specified as a partial assignment of variables, meaning that $vars(\gamma) \subset V$. Generating a random goal state by assigning the unspecified variables with random values often leads to states that satisfy the goal formula, but are unreachable and at odds with human understanding of what a goal state should look like.

An example of this can be observed in the SOKOBAN domain. In the PDDL description of SOKOBAN from IPC 11, there are variables describing the positions of individual boxes and the agent, as well as a special predicate `at-goal` for each box, signifying that the box has been pushed on top of one of the designated goal positions. The goal assignment of a SOKOBAN task is then expressed using only these `at-goal` predicates. As can be seen in Figure 5.1, the fact that the other variables do not appear in the goal can lead to unreachable and intuitively incorrect goal states being generated.

Although the problem of finding only the 'correct' starting points for random backward walks without interference from the user presently remains unsolved, we believe that this approach has a potential advantage compared to simply sampling states by progressing from the initial state.

In the literature, backward walk sampling approaches have been successful in, for example, generating training instances for machine learning approaches to the computation

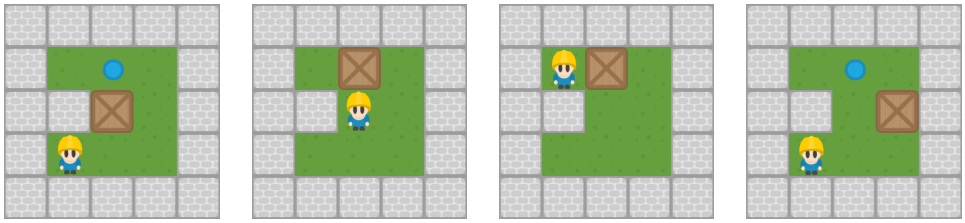


Figure 5.1: Example of the sampling problem in the SOKOBAN *domain. Pictured are (from left ro right): initial state, a reachable goal state, a goal state that is unreachable, but legal according to the rules of Sokoban, and a mutex-violating goal state obtained by assigning $v \notin vars(\gamma)$ randomly.*

of heuristic functions. The studies of these approaches, however, are usually limited to domains where the goal is defined in terms of all or most state variables of the task. For example, in their paper on bootstrap learning, Arfaee et al. (2010) focus on learning heuristics for the n-puzzle, where each state variable has a designated value in the goal assignment.

The reason why it is more desirable to sample states by walking backwards is because in addition to sampling the actual states, one also obtains the (potentially suboptimal) goal distances for each visited state. Using these observed distances, we can reduce the MIP model from the previous section to an LP. Recall that the reason previously described approaches rely on mixed-integer programming is because of the disjunctive descendingness constraint. The constraint states that some successor of state s must be improving. Knowing the goal distances, however, we can directly state which successor must be improving.

This time, S_{sample} contains pairs of states (s, s'), where $s' \in succ(s)$. We then formulate the following LP model over this sample:

$$\min \sum_{(s,s') \in S_{sample}} \alpha_{(s,s')} \tag{5.9}$$

$$\text{s.t.} \quad h(s) - h(s') + \alpha_{(s,s')} \geq 1 \tag{5.10}$$

$$\alpha_{(s,s')} \geq 0 \quad \text{for } (s, s') \in S_{sample} \tag{5.11}$$

The slack variables $\alpha_{(s,s')}$ are necessary to prevent the problem from becoming infeasible whenever mutually contradicting samples are drawn. This can happen if paths $s \to s' \to ... \to s_\star$ and $s' \to s \to ... \to s_\star$ (with $s_\star \in S_\star$) both exist in the transition system of the planing task. In this case, a formulation without slack variables would become infeasible whenever (s, s') and (s', s) are both part of S_{sample}.

This optimization problem is similar to the maximization of GDRC for the selected set of state-pairs. The difference is that the algorithm aims to produce a heuristic that matches the ordering of states based on the observed goal distances, rather than the perfect goal distances.

5.4 Learning Potential Heuristics from Abstractions

The exact computation of DDA heuristics is only feasible for very small planning tasks. As already mentioned, the reasons for this are the usually very large number of states

in the transition system, as well as the size of the feature set, which can be still quite large even after filtering out rare features as described in Section 5.1. Both of these factors scale with the number of state variables (and their domain sizes) and make the problem of computing a DDA heuristic challenging even for modestly-sized planning tasks. Indeed, tasks where the exact DDA heuristic computation would succeed are usually tasks that are easy enough to be solved with simpler heuristics, or even blind search.

In the previous section, we described an attempt to address the state space growth by working with a sample set of states rather than the entire transition system. In this section, we explore the use of abstractions for the purpose of alleviating the issues that arise due to the state- and feature space growth.

In planning, a popular approach for dealing with the aforementioned scaling of the transition system is to compute a heuristic based on an *abstraction* of the planning task. In this thesis, we specifically consider *Pattern Database (PDB)* abstractions (Culberson and Schaeffer, 1998; Edelkamp, 2001). In a PDB-abstracted planning task Π^P, the set of variables V of the original task Π is reduced to a certain subset P, called a *pattern*. Furthermore, all facts referring to variables that are not in P are removed from all partial assignments, including the goal assignment, as well as operator effects and preconditions. A PDB heuristic corresponds to the perfect heuristic of the abstract task Π^P and can be used to guide the search towards a goal in the state space of the concrete task Π.

Analogously to this concept, we introduce *abstract DDA heuristics*. Here, we fist use a pattern selection algorithm to select an abstraction and construct the corresponding abstract task Π^P as described above. Then we apply the naive algorithm from Section 5.2 to this task. This is feasible because the transformation from Π to Π^P produces a smaller task, with fewer states and a small feature set. The resulting heuristic is then used to guide a greedy search algorithm.

Experience shows that PDB heuristics are most successful when many such PDBs capturing different aspects of the original task are combined into a single heuristic function. Because PDB heuristics are inherently admissible, much of the research on combining these heuristics is focused on preserving this property. We refer to Seipp et al. (2017) for a detailed overview of these approaches.

In the context of DDA heuristics and satisficing planning, however, admissibility is neither guaranteed nor necessary. Therefore, we opt to combine multiple abstract DDA heuristic estimates by simple summation.

Chapter 6

Results

We implemented all of the described algorithms in the Fast Downward planning system (Helmert, 2006). In this chapter, we present the results of our empirical evaluation of said algorithms and compare them with existing approaches. Unless stated otherwise, all planners use greedy best-first search (GBFS). We used IBM CPLEX 12.9 to solve the various MIP and LP problems. All calculations were performed on an Intel Xeon Silver 4114 processor running at 2.2 GHz at sciCORE (http://scicore.unibas.ch/) scientific computing center at University of Basel. For each task, the planner was given a time limit of 30 minutes and a memory limit of 7600 MB. As a benchmark set, we used 1816 STRIPS planning tasks from satisficing tracks of all International Planning Competitions (http://ipc.icaps-conference.org). All potential heuristics are of dimension 2. Filtering of systematically generated features (see Section 5.1) is disabled unless stated otherwise. We evaluate the performance of our planners based on the following 3 metrics:

- Coverage: total number of solved tasks
- Cost of the discovered plan
- Number of states expanded during search

6.1 Exact DDA Heuristics

In Section 5.2, we mentioned that the naive algorithm for computing DDA potential heuristics is practically only applicable to planning tasks with very small transition systems. To verify this claim and also to test the quality of the obtained heuristic we compare the slack variable approach from Section 5.2 to the A* search equipped with the *blind* heuristic $h^{\text{blind}}(s) = [\gamma \not\subseteq s]$ as well as GBFS guided by the FF heuristic h^{FF} (Hoffmann and Nebel, 2001). As can be seen in Table 6.1, our configuration h^{DDA} indeed possesses the weakest coverage. In roughly 86% of all cases, the planner stops prematurely due to insufficient memory. Unsurprisingly, all tasks that were solved using h^{DDA} could also be solved optimally using the blind heuristic.

When looking at the solution costs (Figure 6.1) we see that quite a few planning tasks are solved optimally using the exact DDA heuristic. Overall, out of the 157 computed plans 104 (approx. 66%) have the same cost as optimal plans discovered by A* with the blind heuristic. On average, the costs of plans computed using h^{DDA} are greater than the corresponding optimal costs by a factor of 1.21. A cost comparison with the FF heuristic

	h^{DDA}	h^{blind}	h^{FF}
coverage	157	516	1198

Table 6.1: Coverage results for the naively computed DDA potential heuristics compared to the blind and FF heuristics.

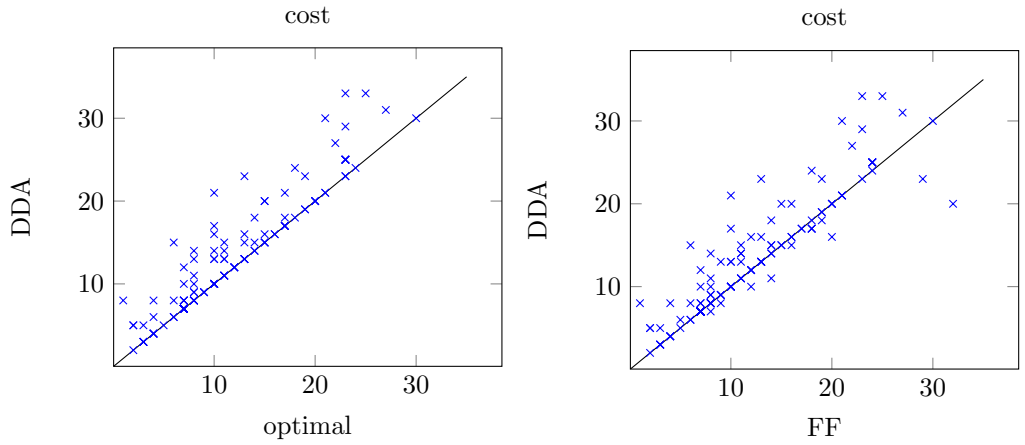


Figure 6.1: Comparison of plan costs of exact DDA planner to the optimal costs as well as the FF heuristic. Not pictured are 9 data points with significantly higher costs.

shows that our planner finds a lower-cost solution for 12 tasks and higher-cost one in 47 cases.

Compared to the expansion numbers of GBFS with the FF heuristic (Figure 6.2), the DDA heuristic causes the search to expand more states in 61 (approx. 39%) tasks. On the other hand, when FF does outperform DDA, it does so by a significant margin. Indeed, in cases where the FF heuristic does worse, it expands on average 8.4 times more states than DDA. In the reverse case, where DDA performs worse, the average number of expansions is approximately 83.7 times greater.

It should be mentioned, that the number of expansions greatly depends on the objective used in the MIP formulation for computing feature weights. As already mentioned in Section 5.2, minimizing the difference between $h(I)$ and the minimum of h would be the ideal objective for the purpose minimizing expansions. The mixed result when compared to the FF heuristic can be explained by the fact that our MIP formulation accepts any DDA heuristic as optimal.

Furthermore, all generated MIP models aim to compute a DDA potential heuristic of dimension 2. In some cases, this dimensionality may be lower than the correlation complexity of the given task, which means that there exists no 2-dimensional DDA potential heuristic. As explained in Section 5.2, we employ slack variables to prevent the MIP from becoming unsolvable in such cases. This, however, leads to potential heuristics that only satisfy DDA properties for a subset of all alive states. Such 'imperfect' DDA heuristics can lead to more expansions.

For example, Gripper is a domain that is known to be of correlation complexity 2 (Seipp et al., 2016). When trying to solve the first Gripper instance from the IPC benchmarks, a 2-dimensional DDA heuristic finds a 15-step solution after 16 expansions, which is expected, since greedy search with a DDA heuristic will never backtrack. A 1-

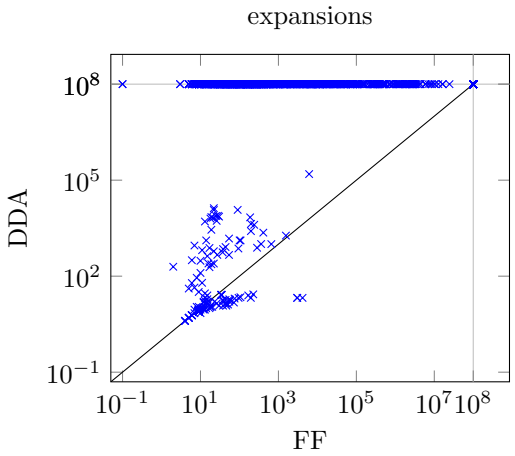


Figure 6.2: Comparison of state expansions of h^{DDA} and the FF heuristic.

dimensional approximation, on the other hand, finds a 13-step solution after expanding 144 states.

Additionally, sometimes the MIP solver exceeds the imposed time and memory limits. Here, the planner proceeds using the heuristic representing the best MIP solution generated until that point. This heuristic also acts as an approximation and leads to a non-ideal expansion count. To illustrate this, we set a 15 second time limit for the MIP solver and demanded that it computes a 2D DDA potential heuristic for the aforementioned GRIPPER instance. In this time, the solver went from an initial solution with an objective value of 254 to a solution whose objective value is 6. It was, however, unable to reach the optimal solution with the objective value of 0. The subsequent search then took 27 expansions to discover a 13-step plan.

6.2 Sample-Based Approximations

One of the major differences between sample-based algorithms from Section 5.3 is the procedure used for sampling. In the following sections, we evaluate the backward- as well as forward-sampling approaches.

Backward Sampling

We first analyse the approach where the sampler starts at some goal state and proceeds with a backward random walk for a random number of steps. Each random walk yields only a single sample data point, consisting of a pair of states $\langle s, s' \rangle$ with $s' \in succ(s)$. As already mentioned, the problem with this approach is in finding an appropriate goal state that will serve as the starting point for the random walk. In the absence of user-provided domain knowledge, the starting point is generated randomly. The problem with this, is that such random goal states may be unreachable. On the other hand, however, generating a new goal state for each random walk allows the sampler to collect data points from many different parts of the transition system.

Thus, the main question when evaluating the backward walk approach is whether it is better to walk backwards from a diverse set of potentially unreachable and mutex-violating

goal states, or to accept domain knowledge from the user in the form of a single goal state that is guaranteed to be reachable.

To address this question, we tested two configurations, namely $h^{\text{bw}}_{\text{rnd}}$, where goal states are generated randomly, and $h^{\text{bw}}_{\text{usr}}$, where the user provides a fixed goal state to the planner. This injection of domain knowledge is simulated in our experiments by solving tasks using the FF heuristic, dumping the discovered goal state, and subsequently providing it to our planner. In cases where the FF planner fails to solve a task, $h^{\text{bw}}_{\text{usr}}$ reverts to $h^{\text{bw}}_{\text{rnd}}$. In both cases, we generate a sample of 1500 state pairs.

Our results show that $h^{\text{bw}}_{\text{rnd}}$ solves 474 planning tasks, while $h^{\text{bw}}_{\text{usr}}$ leads the search to a goal in only 433 cases, which suggests that it is more beneficial to have a diverse pool of goal states rather than a single fixed one that is guaranteed to be reachable. Hence, we will focus on the $h^{\text{bw}}_{\text{rnd}}$ configuration for the remainder of this discussion.

Feature Selection for Backward Sampling

Generally, we see that $h^{\text{bw}}_{\text{rnd}}$ and $h^{\text{bw}}_{\text{usr}}$ already perform significantly better than the attempt to compute exact DDA heuristics. This is due to the fact that setting a low limit on the size of the sample evades the problem of having to deal with the entirety of the state space. The sample-based configurations, however, still succumb to the second scalability issue, namely that of the feature space growth. In the following we evaluate how the feature filtering approach from Section 5.1 can help increase the coverage. Specifically, the approach samples 1000 states as described in Section 5.1 and only keeps those features that appear at least $ft \in \{0, 250, 500, 750\}$ times within the sample. As for the sample for which the LP model will be built, we use samples of size $ssize \in \{125, 250, 500, 1000\}$. In addition to feature filtering, we explored a more aggressive approach: the feature selection algorithm would randomly select 1000 of the systematically generated features and discard all others. We refer to this configuration as flim_{1000}.

Table 6.2 shows that filtering out rare features brings a significant improvement in coverage when compared to the baseline configurations with $ft = 0$. Removing features that do not appear in at least 25-50% of the sample seems to work best. Filtering out features unless they are observed in 75% of the sampled states leads to worse coverage. Nonetheless, even the $ft = 750$ configurations constitute an improvement over the baseline without filtering, thus confirming that exhaustively generating all features results in a performance bottleneck.

All feature filtering configurations, however, are outperformed by flim_{1000}, which is also the only configuration that is able to surpass A* search with the blind heuristic in terms of coverage.

	$ft = 0$	$ft = 250$	$ft = 500$	$ft = 750$	flim_{1000}
$ssize = 125$	469	491	511	493	**538**
$ssize = 250$	477	503	506	495	**560**
$ssize = 500$	479	512	509	498	**575**
$ssize = 1000$	487	508	519	495	**575**

Table 6.2: Coverage for the backward-sampling approach with different minimal numbers of appearances ft needed to include a feature in the final feature set.

	$ft = 0$	$ft = 250$	$ft = 500$	$ft = 750$	flim_{1000}
$ssize = 125$	442	462	465	474	**521**
$ssize = 250$	431	463	454	468	**512**
$ssize = 500$	409	450	445	444	**493**
$ssize = 1000$	381	438	434	438	**490**

Table 6.3: Coverage for the forward-sampling approach h^{fw} with different minimal numbers of appearances ft needed to include a feature in the final feature set.

Forward Sampling

The other sample-based method introduced in Section 5.3 is one we refer to as h^{fw}. Here, the sample set is generated via random walks that originate in the initial state. Because the goal distance is unknown when using this approach, it is not possible to reduce the associated MIP to an LP problem the way it is done with the backward-sampling method. Here, the difference to the exact approach is simply that the MIP is formulated only over the given sample and not the entire reachable and solvable portion of the transition system. Analogously to the backward-sampling method, we evaluate how various sample sizes and feature appearance thresholds influence the coverage.

Similarly to the backward-sampling case, the results in Table 6.3 show that, while the filtering of rare features noticeably improves the performance of the planner, a simple 1000 feature limit leads to a significantly greater increase in coverage. For example, the flim_{1000} configuration with 125 sampled states solves 47 more tasks than the analogous configuration that discards features unless they appear in at least 750 out of 1000 states sampled for feature selection.

Another interesting result is that an increase in the size of the sample leads a *decrease* in coverage. A plausible explanation is that a larger sample implies a larger MIP model that is more difficult to solve. The same observation can also be made in Section 6.3, where we discuss abstract DDA potential heuristics.

Quality of Sample-Based Approximations

The final question we investigate in this section is that of heuristic quality. In the case of the exact algorithm from the Section 6.1, an approximation represents the worst case result that occurs either when the MIP solver exceeds its allocated resources or the correlation complexity of the task is too high. In the case of the sample-based algorithms discussed in this section, however, an approximation of a DDA heuristic is the only possible result. To quantify the effect this has on heuristic quality we, again, compare the number of expansions made by GBFS with the best backward-sampling (resp. best forward-sampling) configuration to the expansions made using heuristics produced by the exact algorithm (see Figure 6.3).

The comparison to $h^{\mathrm{bw}}_{\mathrm{rnd}}$-$\mathit{flim}_{1000}$ with a sample size of 500 shows that this configuration expands fewer states in 33 cases and leads to more expansions in 117 tasks. In the case of forward-sampling, we looked at the flim_{1000} configuration with a sample of size 125. Here, the planner expands fewer states in 30 tasks and more in 123. In conclusion, heuristics produced by the two sample-based approaches are usually of lower quality than heuristics produced by the algorithm that aims to compute an exact DDA heuristic, even though it may also sometimes produce approximations.

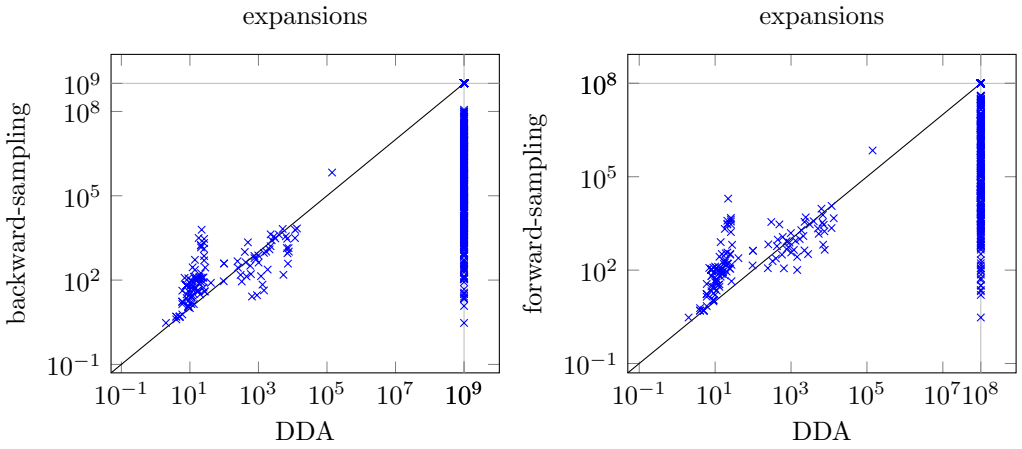


Figure 6.3: Comparison of expansions of the exact DDA planner to the two sampling-based approaches.

6.3 Abstract DDA Heuristics

In Section 5.4, we introduced the concept of abstract DDA potential heuristics, where a DDA heuristic is computed for some abstraction of the original task. In this section, we first discuss the results of using only a single abstract DDA heuristic and then present the results of employing a combination of multiple such heuristics.

Specifically, we begin by evaluating the performance of a planner that automatically generates a single pattern using the greedy algorithm from Fast Downward and then computes the corresponding abstract DDA heuristic. We tested 4 configurations h^{sabs}_k with $k \in \{256, 512, 1024, 2048\}$ as the maximum allowed number of states in the abstract transition system. When evaluating these heuristics, it is most interesting to compare them to PDB heuristics computed from the same abstractions. This comparison is relevant not only because abstract potential DDA heuristics and PDBs share the same underlying concept of abstracting away variables, but also because PDB computation can be viewed as another way to obtain a DDA heuristic. As already mentioned, a PDB heuristic represents the perfect heuristic of the abstract planning task which, in the absence of zero-cost operators, is also DDA.

The coverage results in Table 6.4 show that, among abstract DDA heuristics, coverage is highest for the configuration that uses the smallest abstract state spaces. We believe that a plausible explanation for this is the inherent hardness of finding a DDA heuristic even for small transition systems. Interestingly, the benefit of lowering the hardness of the problem by selecting smaller transition systems seems to outweigh the loss in coverage that is usually associated with coarse abstractions.

With PDBs, on the other hand, the opposite is the case. That is, abstractions with larger transition systems lead to more tasks being solved. We believe that a major reason for this is the fact that our approach to computing DDA heuristics involves solving a MIP, which is an NP-hard problem. PDBs, on the other hand, are computed using Dijkstra's algorithm (Dijkstra, 1959; Sievers et al., 2012). Given a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges, the worst-case complexity of Dijkstra's algorithm is $O(|\mathcal{E}| + |\mathcal{V}| \log |\mathcal{V}|)$ (Barbehenn, 1998). Since $|\mathcal{E}| \leq |\mathcal{V}|^2$ and $|\mathcal{V}|$ is controlled by the

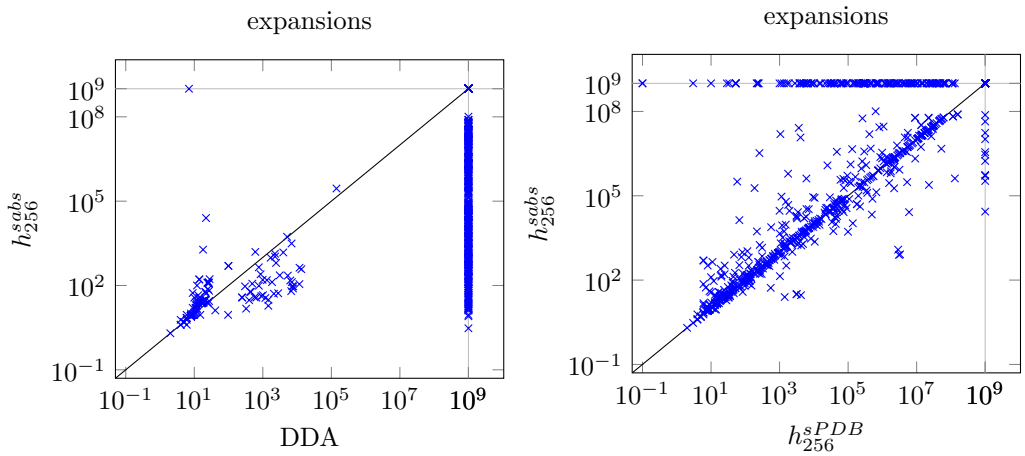


Figure 6.4: Expansion comparison of h^{sabs}_{256} and exact DDA (resp. h^{sPDB}_{256})

	h^{sabs}_k	h^{sPDB}_k
$k = 256$	581	**732**
$k = 512$	561	**747**
$k = 1024$	513	**758**
$k = 2048$	455	**768**

Table 6.4: Coverage results for the various configurations using a single abstraction.

parameter $k \leq 2048$, PDBs can be computed very efficiently in our experiments.

Overall, PDB heuristics show significantly better coverage for the tested transition system sizes. A comparison of states expanded during search (Figure 6.4), however, shows that the h^{sabs}_{256} configuration has generally similar expansions to h^{sPDB}_{256}, which, again, suggests that our planner solves fewer tasks not due to lacking heuristic quality, but rather because the heuristic is expensive to compute. We also see that h^{sabs}_{256} compares favorably to the heuristics produced by the exact DDA algorithm.

For the evaluation of combinations of abstract DDA heuristics we tested 4 configurations h^{mabs}_k with $k \in \{128, 256, 512, 1024\}$ (state-space size limit per abstraction) as well as the special configuration h^{mabs}_{atomic}. Pattern selection for the h^{mabs}_k configurations is performed by the iPDB algorithm (Haslum et al., 2007) under a time limit of 30 seconds. Patterns for the atomic configuration are selected such that they only contain a single variable $v \in vars(\gamma)$. Thus, h^{mabs}_{atomic} works with all distinct patterns containing nothing but a single goal variable.

We compare these configurations to equivalent PDB configurations h^{mPDB}_k. Here, the pattern selection process as well as the method for combining individual heuristics is the same as in the h^{mabs}_k configurations.

The coverage comparison is given in Table 6.5. These results show that abstract potential DDA heuristics vastly outperform the approaches from previous sections, solving roughly twice as many tasks as the sample-based algorithms. The equivalent multiple-PDB approaches, however, perform even better.

When looking at the scalability of the h^{mabs}_k approaches, we see the same result as

	h_k^{mabs}	h_k^{mPDB}
atomic	1028	**1107**
$k = 128$	1005	**1121**
$k = 256$	1005	**1130**
$k = 512$	1005	**1128**
$k = 1024$	999	**1130**

Table 6.5: Coverage results for the various configurations using multiple abstractions. See Table A.3 for a detailed overview.

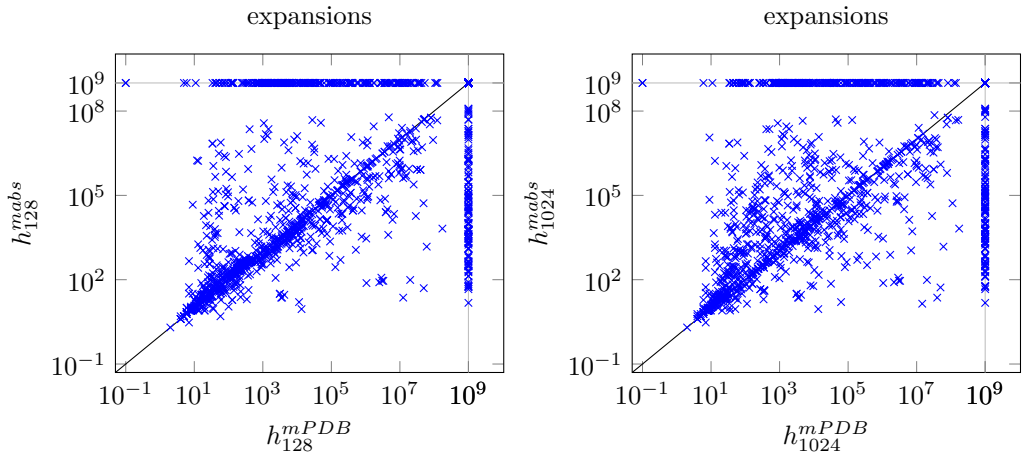


Figure 6.5: Expansion comparison of h_{128}^{mabs} and h_{1024}^{mabs} to the equivalent h^{mPDB} approaches.

with the single-pattern version, that is, better coverage is obtained by choosing abstract tasks with small transition systems. Surprisingly, however, in spite of the less efficient computation, abstract DDA heuristics still perform noticeably better on certain domains, such as OPENSTACKS or TPP.

A look at the expansion comparison between h_k^{mabs} and h_k^{mPDB} for $k \in \{128, 1024\}$ (Figure 6.5), shows that the PDB heuristic often leads to fewer expansions, however, there are quite a few instances where the opposite is the case. Specifically, out of the 871 cases where both, h_{1024}^{mPDB} and h_{1024}^{mabs} could come up with a solution, h_{1024}^{mPDB} expanded fewer search nodes in 436 tasks (approx. 50%). h_{1024}^{mabs} performed better in 262 cases (approx. 30%). When comparing h_{128}^{mPDB} and h_{128}^{mabs} we see that the former expands fewer states in 353 out of 876 cases (40%), while h_{128}^{mabs} performs better in 335 tasks (38%).

When comparing h_{128}^{mabs} to the algorithm that aims to produce exact DDA heuristics (see Figure 6.6), the results are mixed. h_{128}^{mabs} leads to fewer expansions in 54 tasks. The other algorithm, performs better in 56 instances. In the remaining 47 cases, both heuristics result in the same number of expansions.

Compared to the best configuration of $h_{\text{rnd}}^{\text{bw}}$ from the previous section (flim_{1000} with $\mathit{ssize} = 500$), however, h_{128}^{mabs} emerges victorious by solving the overwhelming majority of planning tasks in fewer expansions.

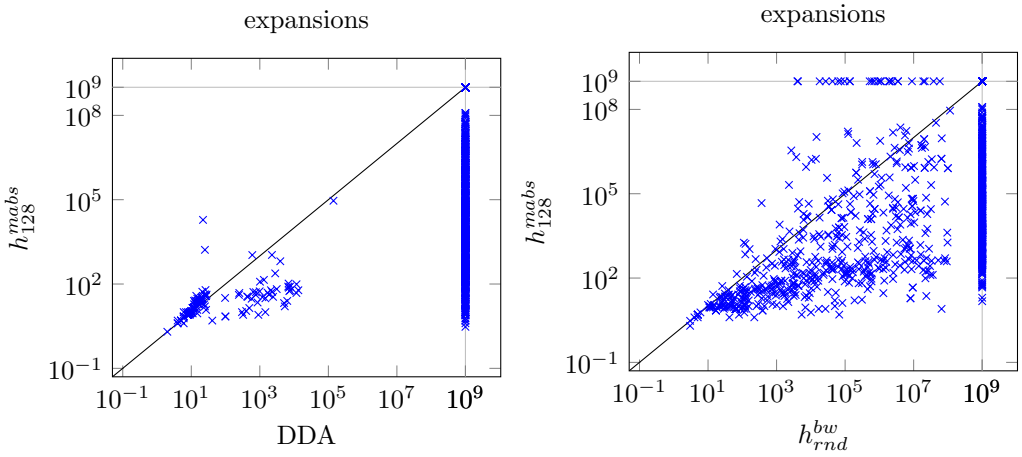


Figure 6.6: Expansion comparison of h_{128}^{abs} to heuristics produced by the exact DDA algorithm and to h_{rnd}^{bw} with a sample size of 500 and a feature set limited to 1000 features.

Chapter 7

Future Work

In this thesis, the strongest of the introduced heuristics in terms of coverage turned out to be the sum of many abstract DDA potential heuristics. A study by Röger and Helmert (2010), however, shows that summation is one of the weakest ways of combining multiple heuristics for satisficing planning. Thus, in future work, it would be interesting to see whether these combined abstract DDA heuristics can be more effective given a more sophisticated way of combining them. This, however, would not address the main bottleneck, that is, the need to solve MIP problems.

Next to the abstract DDA potential heuristics, we believe that the backward sampling approach shows the most promise. Unlike heuristics generated by forward-sampling and the abstract DDA heuristics, we saw the coverage of the backward-sampling approach improve with a larger sample size. Thus, this algorithm has the potential to scale better to larger samples and generate more informative heuristics.

In the thesis, this potential heuristic is learned by forcing an LP solver to choose weights such that the resulting heuristic is able to replicate the correct ranking of points in the sample. The same problem has been the subject of years-long research in the field of *machine-learned ranking (MLR)* (Liu, 2009). Although most MLR research is focused on developing ranking models for information retrieval systems such as web search engines, we believe that parallels can be drawn between the problems of finding a heuristic and finding such a ranking model for retrieved documents.

In multimedia retrieval, the user provides a query that is some vague representation of the desired document and the task of the ranking model is to order available results according to their relevance to the query, whereupon the user greedily chooses the top result. A single step of a greedy search algorithm such as hillclimbing can be viewed in the same setting: out of all immediately available states, the heuristic must recommend the state that is closest to the desired goal.

To our knowledge, most prominent machine learning approaches to heuristic computation such as Bootstrapping (Arfaee et al., 2010) fall into a category known in MLR literature as the *pointwise approaches*. Here, the states used for training come with some cost measure that the heuristic should replicate, i.e. the heuristic is a regression model.

In MLR, this approach is outperformed by the *listwise approach* (Xia et al., 2008), where the goal is to directly optimize the value of some evaluation metric such as Kendall's τ, which is already used by Wilt and Ruml (2015) to quantify the suitability of heuristics for greedy search. Therefore, it would be interesting to investigate whether the listwise approach could lead to better potential heuristics for satisficing planning.

Chapter 8

Conclusion

In this thesis, we have investigated the use of potential heuristics in satisficing planning. Our main focus was on ensuring that these potential heuristics are descending and dead-end avoiding (DDA). The results of our empirical evaluation show that potential heuristics with these properties can guide greedy search towards the goal in a very small number of expansions.

However, our theoretical and empirical analysis also shows that DDA heuristics are generally prohibitively expensive to compute even for small planning tasks. Specifically, we have proven that the computation of DDA heuristics is a PSPACE-complete problem and, thus as hard as planning itself.

Following this result, we investigated the use of various approaches to approximating DDA heuristics. These approaches include stopping heuristic computation prematurely once time or memory limits have been reached, attempting to ensure the DDA set of properties on a randomly selected sample of states, and using abstractions to downscale planning tasks to sizes that can be handled by the exact DDA computation routine.

Our experiments show that, out of all introduced configurations, the sum of many abstract DDA heuristics is by far the most powerful approach. Nonetheless, this approach still falls short of the performance and scalability offered by equivalent PDB heuristics. In terms of heuristic quality, however the two heuristics are comparable. The empirical results suggest that the performance of the abstract DDA heuristic is mainly hampered by the need to solve MIP problems.

Bibliography

Shahab Jabbari Arfaee, Sandra Zilles, and Robert C. Holte. Bootstrap learning of heuristic functions. In *Proceedings of the Third Annual Symposium on Combinatorial Search*, pages 52–60, 2010.

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

Michael Barbehenn. A note on the complexity of Dijkstra's Algorithm for graphs with weighted vertices. *IEEE Transactions on Computers*, 47(2):263, 1998.

Blai Bonet. An admissible heuristic for SAS^+ planning obtained from the state equation. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 2268–2274, 2013.

Michael Buro. From simple features to sophisticated evaluation functions. In *Computers and Games, First International Conference*, pages 126–145, 1998.

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

Augusto B. Corrêa and Florian Pommerening. An empirical study of perfect potential heuristics. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling*, pages 114–118, 2019.

Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

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

Stefan Edelkamp. Planning with pattern databases. In *Sixth European Conference on Planning*, pages 13–24, 2001.

Andreas Ferenczi. Berechnung von Potential-Heuristiken basierend auf Pattern–Datenbanken, 2016. Bachelor's Thesis.

Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

Guillem Francès, Augusto B. Corrêa, Cedric Geissmann, and Florian Pommerening. Generalized potential heuristics for classical planning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 5554–5561, 2019.

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

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

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

Jörg Hoffmann. Local search topology in planning benchmarks: A theoretical analysis. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*, pages 92–100, 2002.

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

Maurice George Kendall. *Rank correlation methods.* Charles Griffin & Co. Ltd., 1948.

Hans Kleine Büning and Uwe Bubeck. Theory of quantified boolean formulas. In *Handbook of Satisfiability*, pages 735–760. 2009.

Donald E. Knuth. Two notes on notation. *The American Mathematical Monthly*, 99(5): 403–422, 1992.

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

Richard E. Korf, Michael Reid, and Stefan Edelkamp. Time complexity of iterative-deepening-A*. *Artificial Intelligence*, 129(1-2):199–218, 2001.

Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

Albert R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Thirteenth Annual Symposium on Switching and Automata Theory*, pages 125–129, 1972.

Florian Pommerening and Malte Helmert. A normal form for classical planning tasks. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling*, pages 188–192, 2015.

Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From non-negative to general operator cost partitioning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 3335–3341, 2015.

Florian Pommerening, Malte Helmert, and Blai Bonet. Higher-dimensional potential heuristics for optimal classical planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 3636–3643, 2017.

Gabriele Röger and Malte Helmert. The more, the merrier: Combining heuristic estimators for satisficing planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*, pages 246–249, 2010.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence – A Modern Approach*. Prentice Hall, 1995.

Jendrik Seipp, Florian Pommerening, and Malte Helmert. New optimization functions for potential heuristics. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling*, pages 193–201, 2015.

Jendrik Seipp, Florian Pommerening, Gabriele Röger, and Malte Helmert. Correlation complexity of classical planning domains. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 3242–3250, 2016.

Jendrik Seipp, Thomas Keller, and Malte Helmert. A comparison of cost partitioning algorithms for optimal classical planning. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling*, pages 259–268, 2017.

Silvan Sievers, Manuela Ortlieb, and Malte Helmert. Efficient implementation of pattern database heuristics for classical planning. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search*, pages 105–111, 2012.

Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1): 1–22, 1976.

Christopher M. Wilt and Wheeler Ruml. Building a heuristic for greedy search. In *Eighth Annual Symposium on Combinatorial Search*, pages 131–140, 2015.

Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: theory and algorithm. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference*, pages 1192–1199, 2008.

Appendix A

Detailed Coverage Results

Coverage	**blind**	**FF**	h^{DDA}
agricola-sat18-strips (20)	1	**10**	0
airport (50)	23	**36**	1
barman-sat11-strips (20)	0	**6**	0
barman-sat14-strips (20)	0	**11**	0
blocks (35)	21	**35**	9
childsnack-sat14-strips (20)	0	**1**	0
data-network-sat18-strips (20)	0	**4**	0
depot (22)	6	**17**	1
driverlog (20)	8	**18**	1
elevators-sat08-strips (30)	2	**11**	0
elevators-sat11-strips (20)	**0**	**0**	**0**
floortile-sat11-strips (20)	0	**8**	0
floortile-sat14-strips (20)	0	**2**	0
freecell (80)	20	**79**	0
ged-sat14-strips (20)	**0**	**0**	**0**
grid (5)	2	**4**	0
gripper (20)	8	**20**	3
hiking-sat14-strips (20)	3	**20**	0
logistics00 (28)	11	**28**	3
logistics98 (35)	2	**29**	0
miconic (150)	55	**150**	25
movie (30)	**30**	**30**	**30**
mprime (35)	20	**32**	0
mystery (30)	15	**17**	1
nomystery-sat11-strips (20)	4	**10**	1
openstacks-sat08-strips (30)	**13**	6	3
openstacks-sat11-strips (20)	**0**	**0**	**0**
openstacks-sat14-strips (20)	**0**	**0**	**0**
openstacks-strips (30)	7	**28**	5
organic-synthesis-sat18-strips (20)	**3**	**3**	0
organic-synthesis-split-sat18-strips (20)	4	**14**	0
parcprinter-08-strips (30)	11	**23**	3
parcprinter-sat11-strips (20)	0	**7**	0
parking-sat11-strips (20)	0	**20**	0
parking-sat14-strips (20)	0	**14**	0

Coverage	**blind**	**FF**	h^{DDA}
pathways-noneg (30)	4	**10**	1
pegsol-08-strips (30)	28	**30**	2
pegsol-sat11-strips (20)	18	**20**	0
pipesworld-notankage (50)	18	**31**	0
pipesworld-tankage (50)	13	**23**	0
psr-small (50)	**50**	**50**	38
rovers (40)	6	**26**	4
satellite (36)	6	**29**	2
scanalyzer-08-strips (30)	12	**28**	3
scanalyzer-sat11-strips (20)	4	**18**	0
snake-sat18-strips (20)	3	**5**	0
sokoban-sat08-strips (30)	18	**28**	0
sokoban-sat11-strips (20)	8	**18**	0
spider-sat18-strips (20)	1	**18**	0
storage (30)	15	**19**	7
termes-sat18-strips (20)	0	**14**	0
tetris-sat14-strips (20)	0	**9**	0
thoughtful-sat14-strips (20)	5	**8**	0
tidybot-sat11-strips (20)	4	**16**	0
tpp (30)	6	**23**	4
transport-sat08-strips (30)	6	**13**	3
transport-sat11-strips (20)	**0**	**0**	**0**
transport-sat14-strips (20)	**0**	**0**	**0**
trucks-strips (30)	7	**15**	2
visitall-sat11-strips (20)	0	**3**	0
visitall-sat14-strips (20)	**0**	**0**	**0**
woodworking-sat08-strips (30)	6	**29**	2
woodworking-sat11-strips (20)	1	**14**	1
zenotravel (20)	8	**20**	2
Sum (1816)	516	**1210**	157

Table A.1: Coverage comparison between h^{DDA}, A equipped with a blind heuristic, and FF.*

Coverage	h^{sabs}_{256}	h^{sabs}_{512}	h^{sabs}_{1024}	h^{sabs}_{2048}	h^{sPDB}_{256}	h^{sPDB}_{512}	h^{sPDB}_{1024}	h^{sPDB}_{2048}
agricola-sat18-strips (20)	8	8	5	0	**12**	**12**	**12**	**12**
airport (50)	27	26	26	17	27	**29**	**29**	**29**
barman-sat11-strips (20)	0	0	0	0	**4**	**4**	**4**	0
barman-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
blocks (35)	20	16	16	15	23	23	23	**24**
childsnack-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
data-network-sat18-strips (20)	**1**	**1**	0	0	**1**	**1**	**1**	**1**
depot (22)	**7**	**7**	6	6	**7**	**7**	**7**	**7**
driverlog (20)	8	8	3	2	**11**	**11**	**11**	**11**
elevators-sat08-strips (30)	**3**	**3**	2	2	2	2	2	2
elevators-sat11-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
floortile-sat11-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
floortile-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
freecell (80)	40	40	32	26	63	63	65	**72**
ged-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
grid (5)	**3**	**3**	**3**	2	**3**	**3**	**3**	**3**
gripper (20)	10	10	6	7	**11**	**11**	**11**	**11**
hiking-sat14-strips (20)	12	9	7	5	16	**20**	**20**	**20**
logistics00 (28)	15	8	3	3	15	15	**16**	**16**
logistics98 (35)	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
miconic (150)	75	78	79	81	75	80	84	**85**
movie (30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
mprime (35)	**24**	19	16	12	**24**	**24**	**24**	**24**
mystery (30)	14	12	10	6	**17**	**17**	**17**	**17**
nomystery-sat11-strips (20)	**6**	**6**	4	4	**6**	**6**	**6**	**6**
openstacks-sat08-strips (30)	**12**	**12**	**12**	**12**	6	6	6	6
openstacks-sat11-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
openstacks-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
openstacks-strips (30)	14	14	15	15	15	15	**16**	**16**
organic-synthesis-sat18-strips (20)	1	1	2	2	**3**	**3**	**3**	**3**
organic-synthesis-split-sat18-strips (20)	2	3	2	1	**5**	**5**	**5**	**5**
parcprinter-08-strips (30)	17	17	17	17	17	**18**	**18**	**18**
parcprinter-sat11-strips (20)	2	**3**	**3**	**3**	2	**3**	**3**	**3**
parking-sat11-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
parking-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
pathways-noneg (30)	5	5	5	**6**	5	5	5	**6**
pegsol-08-strips (30)	17	21	24	19	**28**	**28**	**28**	**28**
pegsol-sat11-strips (20)	4	10	13	9	**18**	**18**	**18**	**18**
pipesworld-notankage (50)	9	7	10	12	**40**	**40**	**40**	**40**
pipesworld-tankage (50)	8	5	6	4	**15**	13	14	14
psr-small (50)	**50**	48	46	45	**50**	**50**	**50**	**50**
rovers (40)	17	17	17	12	17	18	18	**19**
satellite (36)	9	10	9	**12**	9	10	11	**12**
scanalyzer-08-strips (30)	11	11	10	8	**15**	**15**	**15**	**15**
scanalyzer-sat11-strips (20)	3	3	2	2	**6**	**6**	**6**	**6**
snake-sat18-strips (20)	0	2	1	0	4	**5**	**5**	4
sokoban-sat08-strips (30)	19	16	8	3	24	24	24	**25**
sokoban-sat11-strips (20)	11	10	5	1	**15**	**15**	**15**	**15**
spider-sat18-strips (20)	**4**	2	2	0	1	1	1	1
storage (30)	14	13	14	13	**19**	**19**	**19**	**19**
termes-sat18-strips (20)	0	0	0	0	0	0	0	**1**
tetris-sat14-strips (20)	0	0	0	0	**1**	**1**	0	0
thoughtful-sat14-strips (20)	5	4	1	2	5	5	5	**6**

Coverage	h^{sabs}_{256}	h^{sabs}_{512}	h^{sabs}_{1024}	h^{sabs}_{2048}	h^{sPDB}_{256}	h^{sPDB}_{512}	h^{sPDB}_{1024}	h^{sPDB}_{2048}
tidybot-sat11-strips (20)	1	4	2	2	19	19	**20**	**20**
tpp (30)	11	11	**12**	**12**	11	11	**12**	**12**
transport-sat08-strips (30)	6	6	6	6	6	**7**	**7**	**7**
transport-sat11-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
transport-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
trucks-strips (30)	**9**	**9**	**9**	8	**9**	**9**	**9**	**9**
visitall-sat11-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
visitall-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
woodworking-sat08-strips (30)	**6**	5	5	4	**6**	**6**	**6**	**6**
woodworking-sat11-strips (20)	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
zenotravel (20)	7	4	3	3	**10**	**10**	**10**	**10**
Sum (1816)	581	561	513	455	732	747	758	**768**

Table A.2: Coverage of single abstract DDA heuristics and corresponding PDBs.

Coverage	h^{mPDB}_{atomic}	h^{mPDB}_{128}	h^{mPDB}_{256}	h^{mPDB}_{512}	h^{mPDB}_{1024}	h^{mabs}_{atomic}	h^{mabs}_{128}	h^{mabs}_{256}	h^{mabs}_{512}	h^{mabs}_{1024}
agricola18 (20)	10	14	15	**16**	**16**	9	13	15	15	15
airport (50)	29	29	29	29	28	32	32	33	34	**35**
barman11 (20)	4	14	14	14	**16**	4	0	0	0	0
barman14 (20)	0	3	3	3	**4**	0	0	0	0	0
blocks (35)	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**
childsnack14 (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
data-network18 (20)	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
depot (22)	12	12	**14**	**14**	**14**	9	10	10	9	10
driverlog (20)	16	16	**17**	16	**17**	13	12	13	13	12
elevators08 (30)	2	7	**8**	**8**	6	7	**8**	**8**	**8**	7
elevators11 (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
floortile11 (20)	1	1	0	0	**2**	**2**	**2**	**2**	**2**	**2**
floortile14 (20)	0	0	1	1	0	**2**	**2**	**2**	**2**	**2**
freecell (80)	75	76	76	76	76	**78**	73	77	77	77
ged14 (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
grid (5)	**4**	**4**	**4**	**4**	**4**	3	3	3	3	3
gripper (20)	**20**	**20**	**20**	**20**	**20**	11	11	11	11	11
hiking14 (20)	**20**	**20**	**20**	**20**	**20**	**20**	14	13	12	12
logistics00 (28)	**28**	**28**	**28**	**28**	**28**	22	22	22	22	22
logistics98 (35)	**35**	**35**	**35**	**35**	**35**	8	8	7	7	7
miconic (150)	**150**	**150**	**150**	**150**	**150**	**150**	**150**	**150**	**150**	**150**
movie (30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
mprime (35)	**25**	**25**	**25**	**25**	**25**	**25**	23	20	22	23
mystery (30)	**17**	**17**	**17**	**17**	16	16	14	16	15	13
nomystery11 (20)	**14**	12	7	7	7	13	**14**	12	11	8
openstacks08 (30)	6	6	6	6	6	**30**	**30**	**30**	**30**	**30**
openstacks11 (20)	0	0	0	0	0	**16**	**16**	**16**	**16**	**16**
openstacks14 (20)	0	0	0	0	0	**11**	10	10	10	10
openstacks (30)	26	**27**	**27**	**27**	**27**	26	**27**	**27**	**27**	**27**
organic-synthesis18 (20)	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
organic-synthesis-split18 (20)	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
parcprinter-08 (30)	12	12	12	12	12	**22**	**22**	**22**	**22**	**22**
parcprinter11 (20)	0	0	0	0	0	12	**13**	**13**	**13**	**13**
parking11 (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Coverage	h^{mPDB}_{atomic}	h^{mPDB}_{128}	h^{mPDB}_{256}	h^{mPDB}_{512}	h^{mPDB}_{1024}	h^{mabs}_{atomic}	h^{mabs}_{128}	h^{mabs}_{256}	h^{mabs}_{512}	h^{mabs}_{1024}
parking14 (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
pathways-noneg (30)	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
pegsol-08 (30)	28	29	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
pegsol11 (20)	18	19	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
pipesworld-notankage (50)	**40**	39	39	38	38	39	37	37	37	37
pipesworld-tankage (50)	16	**17**	**17**	15	14	**17**	15	15	16	16
psr-small (50)	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**
rovers (40)	21	**29**	**29**	**29**	**29**	21	28	26	24	24
satellite (36)	**18**	**18**	**18**	**18**	**18**	16	14	14	15	14
scanalyzer-08 (30)	**30**	28	29	**30**	**30**	14	14	14	14	14
scanalyzer11 (20)	**20**	18	**20**	**20**	**20**	5	5	5	5	5
snake18 (20)	5	**7**	**7**	**7**	**7**	5	6	5	6	5
sokoban08 (30)	**25**	17	17	17	17	23	22	22	22	22
sokoban11 (20)	**15**	7	7	7	7	14	12	12	12	12
spider18 (20)	**9**	**9**	**9**	**9**	**9**	7	**9**	**9**	**9**	8
storage (30)	**19**	18	18	18	18	18	16	16	16	16
termes18 (20)	10	11	11	11	**12**	5	5	5	5	5
tetris14 (20)	**20**	18	19	19	19	**20**	17	17	17	16
thoughtful14 (20)	5	9	12	12	**14**	5	7	7	7	7
tidybot11 (20)	**19**	**19**	**19**	**19**	**19**	**19**	14	14	14	15
tpp (30)	9	9	9	9	9	**29**	**29**	**29**	**29**	**29**
transport08 (30)	**30**	**30**	**30**	**30**	**30**	12	12	12	12	12
transport11 (20)	**20**	**20**	**20**	**20**	**20**	0	0	0	0	0
transport14 (20)	**17**	**17**	**17**	**17**	**17**	0	0	0	0	0
trucks (30)	**9**	**9**	**9**	**9**	8	**9**	7	7	7	7
visitall11 (20)	**20**	**20**	**20**	**20**	**20**	5	4	4	5	5
visitall14 (20)	**20**	**20**	**20**	**20**	**20**	0	0	0	0	0
woodworking08 (30)	**11**	9	9	9	9	10	10	10	10	10
woodworking11 (20)	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
zenotravel (20)	**16**	**16**	**16**	**16**	**16**	13	13	13	12	13
Sum (1816)	1107	1121	**1130**	1128	**1130**	1028	1005	1005	1005	999

Table A.3: Coverage of configurations using multiple abstract DDA heuristics and compared to analogous PDB configurations.

Coverage	h^{DDA}	$h^{\mathbf{bw}}_{\mathbf{rnd}}$**-flim1k(ssize=500)**	h^{fw}**-flim1k(ssize=125)**	h^{sabs}_{256}	h^{mabs}_{atomic}
agricola-sat18-strips (20)	0	11	**13**	8	9
airport (50)	1	18	22	27	**32**
barman-sat11-strips (20)	0	0	0	0	**4**
barman-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**
blocks (35)	9	21	21	20	**35**
childsnack-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**
data-network-sat18-strips (20)	0	0	0	**1**	**1**
depot (22)	1	6	5	7	**9**
driverlog (20)	1	6	7	8	**13**
elevators-sat08-strips (30)	0	3	2	3	**7**
elevators-sat11-strips (20)	**0**	**0**	**0**	**0**	**0**
floortile-sat11-strips (20)	0	0	0	0	**2**
floortile-sat14-strips (20)	0	1	0	0	**2**
freecell (80)	0	25	19	40	**78**
ged-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**
grid (5)	0	2	0	**3**	**3**
gripper (20)	3	9	9	10	**11**
hiking-sat14-strips (20)	0	7	11	12	**20**
logistics00 (28)	3	10	10	15	**22**
logistics98 (35)	0	3	2	3	**8**
miconic (150)	25	80	63	75	**150**
movie (30)	**30**	**30**	**30**	**30**	**30**
mprime (35)	0	20	18	24	**25**
mystery (30)	1	10	7	14	**16**
nomystery-sat11-strips (20)	1	6	4	6	**13**
openstacks-sat08-strips (30)	3	8	6	12	**30**
openstacks-sat11-strips (20)	0	0	0	0	**16**
openstacks-sat14-strips (20)	0	0	0	0	**11**
openstacks-strips (30)	5	7	7	14	**26**
organic-synthesis-sat18-strips (20)	0	**3**	2	1	**3**
organic-synthesis-split-sat18-strips (20)	0	1	3	2	**5**
parcprinter-08-strips (30)	3	11	10	17	**22**
parcprinter-sat11-strips (20)	0	0	0	2	**12**
parking-sat11-strips (20)	**0**	**0**	**0**	**0**	**0**
parking-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**
pathways-noneg (30)	1	4	4	5	**6**
pegsol-08-strips (30)	2	**30**	29	17	**30**
pegsol-sat11-strips (20)	0	**20**	19	4	**20**
pipesworld-notankage (50)	0	21	14	9	**39**
pipesworld-tankage (50)	0	15	13	8	**17**
psr-small (50)	38	**50**	**50**	**50**	**50**
rovers (40)	4	15	8	17	**21**
satellite (36)	2	8	7	9	**16**
scanalyzer-08-strips (30)	3	12	12	11	**14**
scanalyzer-sat11-strips (20)	0	4	4	3	**5**
snake-sat18-strips (20)	0	**18**	5	0	5
sokoban-sat08-strips (30)	0	15	18	19	**23**
sokoban-sat11-strips (20)	0	6	9	11	**14**
spider-sat18-strips (20)	0	2	1	4	**7**
storage (30)	7	17	16	14	**18**
termes-sat18-strips (20)	0	0	0	0	**5**
tetris-sat14-strips (20)	0	0	1	0	**20**
thoughtful-sat14-strips (20)	0	3	3	**5**	**5**

Coverage	h^{DDA}	$h^{\mathbf{bw}}_{\mathbf{rnd}}$**-flim1k(ssize=500)**	h^{fw}**-flim1k(ssize=125)**	h^{sabs}_{256}	h^{mabs}_{atomic}
tidybot-sat11-strips (20)	0	0	3	1	**19**
tpp (30)	4	8	7	11	**29**
transport-sat08-strips (30)	3	6	6	6	**12**
transport-sat11-strips (20)	**0**	**0**	**0**	**0**	**0**
transport-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**
trucks-strips (30)	2	8	7	**9**	**9**
visitall-sat11-strips (20)	0	0	0	0	**5**
visitall-sat14-strips (20)	**0**	**0**	**0**	**0**	**0**
woodworking-sat08-strips (30)	2	5	5	6	**10**
woodworking-sat11-strips (20)	**1**	**1**	**1**	**1**	**1**
zenotravel (20)	2	9	8	7	**13**
Sum (1816)	157	575	521	581	**1028**

Table A.4: Coverage comparison of best configurations of each approach.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Alexander Rovner

Matriculation number — Matrikelnummer
2015-050-289

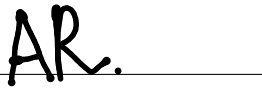
Title of work — Titel der Arbeit
Potential Heuristics for Satisficing Planning

Type of work — Typ der Arbeit
Master's Thesis

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, February 4, 2020



Signature — Unterschrift