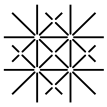


University of Basel

Evaluation Of Post-Hoc Optimization Constraints Under Altered Cost Functions

Master's Thesis

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

Examiner: Dr. Gabriele Röger
Supervisor: Dr. Florian Pommerening

Andreas Thüring
a.thuering@unibas.ch
09-724-162

February 6, 2019

Acknowledgments

I thank my supervisor Dr. Florian Pommerening for sharing his feedback and supporting me during the course of writing this thesis. I also thank Dr. Gabriele Röger for allowing me to write this thesis at the Artificial Intelligence group. Finally, I thank the sciCORE (https://scicore.unibas.ch/) scientific computing center at University of Basel for letting me use their resources to perform experiments.

Abstract

The operator-counting framework (Pommerening et al., 2014) is a framework in classical planning for heuristics that are based on linear programming. The operator-counting framework covers several kinds of state-of-the-art linear programming heuristics, among them the post-hoc optimization heuristic (Pommerening et al., 2013). In this thesis we will use post-hoc optimization constraints and evaluate them under altered cost functions instead of the original cost function of the planning task. We show that such *cost-altered post-hoc optimization constraints* are also covered by the operator-counting framework and that it is possible to achieve improved heuristic estimates with them, compared with post-hoc optimization constraints under the original cost function. In our experiments we have not been able found to achieve improved problem coverage, as we were not able to find a method for generating favorable cost functions that work well in all domains.

Table of Contents

1 Introduction

The goal of classical planning, a field in Artificial Intelligence research, is to find a plan given a formulation of a planning task. A plan is a sequence of actions that begins in the initial state and ends in a goal state, as defined by the planning task. Since exhaustively searching the state space for a plan is infeasible for larger planning tasks, heuristic search algorithms employ a so-called heuristic function to estimate the distance from a specific state to a goal condition in order to reduce the number of states that are required to explore to find a solution. If the heuristic search algorithm A^* (Hart et al., 1968) is used in conjunction with an admissible heuristic, it finds an optimal plan: a plan with minimal total action costs among all plans. Finding an optimal plan is the occupation of optimal planning, which we will focus on in this thesis. Satisficing planning finds a plan of arbitrary length and will not be discussed here.

Pattern databases (Edelkamp, 2001) derive an admissible heuristic from the cost of an optimal plan in an abstraction of the original problem. An abstraction, or pattern, is a simplified representation that discards some aspects of the original planning task. There exist methods that can retrieve the stored goal distances efficiently, but since the goal distance needs to be stored for every abstract state that is considered in a pattern, the size of a single pattern is typically kept rather small due to memory constraints. This in turn limits the heuristic information that a single pattern can provide.

It is possible to render pattern database heuristics more informative by combining the heuristic estimates of multiple patterns into a single heuristic, but simply summing up the individual pattern database heuristic estimates is not guaranteed to be admissible. As noted by Edelkamp (2001), the maximum of multiple pattern database heuristics is admissible and dominates a single pattern database heuristic. Edelkamp (2001) also introduce the notion of *disjoint pattern databases*, of which the heuristic functions can be summed up admissibly for stronger heuristic estimates than taking their maximum. Haslum et al. (2007) extend this concept of additivity with the *canonical heuristic* that finds a maximal additive subset out of a given set of patterns. *Cost partitioning* (Katz and Domshlak, 2010) adapts the cost functions of abstract tasks to derive additive pattern sets from arbitrary sets of abstractions. Using linear programs (LPs) to compute such an *optimal cost partitioning* provides a very strong heuristic, yet Pommerening et al. (2013) note that the LPs involved

in the heuristic computation for every state grow prohibitively large for realistic problem sizes. While optimal cost partitioning provides a very strong approach to the admissible combination of abstraction heuristics in theory, its performance implications often render it infeasible to use in practice.

Pommerening et al. (2014) provide a framework for admissible integer/linear programming heuristics called the *operator-counting framework*. This framework provides a predefined structure for the optimization function and for the so-called *operator-counting constraints* of the linear program, while leaving it up to the specific implementation where the information for the constraints is derived from. Pommerening et al. (2014) show that abovementioned optimal cost partitioning among other established as well as novel techniques can be expressed as operator-counting constraints. The operator-counting framework enables us to compare different linear programming heuristics from a theoretical perspective via their constraints and to easily combine them in a single linear program.

The *post-hoc optimization heuristic* (Pommerening et al., 2013) is a heuristic based on linear programming that dominates the canonical heuristic and achieves performance comparable to other state-of-the-art linear programming heuristics. Pommerening et al. (2014) show that the post-hoc optimization heuristic can be represented in the operator-counting framework by only a single operator-counting constraint, which are far fewer than are required for an optimal cost partitioning. A further optimization is that in each constraint the post-hoc optimization heuristic ignores operators that are deemed to be non-contributing for the corresponding heuristic.

In this thesis we will evaluate post-hoc optimization constraints under cost functions that are different from the original cost function of the planning task. In chapter 2 there is an introduction to SAS^+ planning and the operator-counting framework with a focus on post-hoc optimization constraints, which form the basis of our extension. In chapter 3 we introduce cost-altered post-hoc optimization constraints and we show that they are operator-counting constraints. We then show that cost-altered post-hoc optimization constraints dominate regular post-hoc optimization constraints in some planning tasks, by providing a constructed example. A discussion of experiment results that we obtained by running an implementation of our method in the planning system Fast Downward (Helmert, 2011) is found in chapter 4: throughout a set of established planning task domains, we compare our implementation with regular post-hoc optimization constraints and other types of constraint sets that are included in the operator-counting framework as provided by Fast Downward.

2
Background

In this section we provide a description of classical planning with finite-domain state variables in SAS$^+$ as far as we will employ it throughout our thesis, as well as an explanation of the operator-counting framework. We will then focus specifically on post-hoc optimization constraints, which form the basis of our extension that is detailed in chapter 3.

2.1 Classical Planning

In this thesis we consider planning tasks that are formulated in SAS$^+$(Bäckström and Nebel, 1995).

Definition 1 (planning task)**.** A planning task is a tuple $\Pi = \langle V, O, s_0, s_*, \mathit{cost} \rangle$ where

- V is a set of finite-domain variables. Each variable $v \in V$ has an associated finite domain $\mathit{dom}(v)$. A variable assignment is a function $f : V' \to \bigcup_{v \in V'} \mathit{dom}(v)$ where $f(v) \in \mathit{dom}(v)$ for every $v \in V' \subseteq V$. The set of variables V' for which f is defined is denoted as $\mathit{vars}(f)$. A *state* is a total variable assignment, i.e. $\mathit{vars}(f) = V$. In a *partial state* a subset of variables $\mathit{vars}(f) \subseteq V$ is defined. We write $s(v) \in \mathit{dom}(v)$ for the assignment of a particular variable $v \in V$ in (partial) state s. A state s is said to be *consistent* with partial state s', written $s' \sqsubseteq s$, if $s(v) = s'(v)$ for all $v \in \mathit{vars}(s')$. The set of all states is denoted as $S(\Pi)$.

- O is a set of operators where each *operator* $o \in O$ is a tuple $o = \langle \mathit{pre}(o), \mathit{eff}(o) \rangle$. The partial state $\mathit{pre}(o)$ is called the *precondition* of o, and the partial state $\mathit{eff}(o)$ is called the *effect* of operator o. An operator o is *applicable* in state s if $\mathit{pre}(o) \sqsubseteq s$. The *successor* $s' = s[o]$ under the application of applicable operator o in state s is the state defined for all $v \in V$ as

$$s'(v) = \begin{cases} \mathit{eff}(o)(v) & \text{if } v \in \mathit{vars}(\mathit{eff}(o)), \\ s(v) & \text{otherwise.} \end{cases}$$

- s_0 is the *initial state*.

- s_* is a partial state called the *goal condition*.

- *cost* is a cost function, that is, a total function $cost : O \to \mathbb{N}_0$ that assigns each operator $o \in O$ a non-negative cost.

A sequence of operators $\pi = \{o_1, \ldots, o_k\}$ is called *applicable* in state s if there is a sequence of states $\{s_0, \ldots, s_k\}$ such that operator o_i is applicable in state s_{i-1} and $s_i = s_{i-1}[o_i]$ for all $i = 1, \ldots, k$. We write $s_k = s[\pi]$ for the last state in the sequence. An *s-plan* is an applicable sequence of operators starting in state s where $s[\pi]$ is a *goal state*: a state consistent with the goal condition. The *cost* of an s-plan $cost(\pi) = \sum_{i=1}^{k} cost(o_i)$ is the sum of all its associated action costs. An *optimal s-plan* is an s-plan with minimal costs among all s-plans. An s_0-plan starting in the initial state s_0 is often just called a *plan*.

2.2 Heuristic Search

Informed search or *heuristic search* employs a function called the *heuristic function* that estimates the cost of an s-plan from state $s \in S(\Pi)$ to a state that is consistent with the goal condition. The function $h^*(s) = cost(\pi^*)$ for an optimal s-plan π^* is called the *perfect heuristic*, denoting the cost of the shortest possible s-plan to a goal state.

Definition 2 (properties of a heuristic function)**.** Let $\Pi = \langle V, O, s_0, s_*, cost \rangle$ be a planning task. The heuristic function $h : S(\Pi) \times cost \to \mathbb{N}_0 \cup \{\infty\}$ is

- *admissible* under *cost* if $h(s, cost) \leq h^*(s, cost)$ for all $s \in S(\Pi)$, i.e. h never overestimates the true cost of an optimal s-plan,
- *goal-aware* under *cost* if $h(s, cost) = 0$ for all $s \in S(\Pi)$ that are consistent with the goal condition s_*.
- *consistent* under *cost* if $h(s, cost) \leq cost(o) + h(s[o], cost)$ for every state $s \in S(\Pi)$ and operator $o \in O$ that is applicable in s.

A heuristic that is goal-aware and consistent is also admissible (Russell and Norvig, 1995).

Optimal planning is concerned with finding an optimal plan for a given planning task Π. The A^* search algorithm (Hart et al., 1968) is a heuristic search algorithm that, in conjunction with an admissible heuristic, returns an optimal plan for a planning task Π, if such a plan exists.

2.3 Transition Systems and Abstraction Heuristics

It is often useful to interpret a planning task as a directed, labelled graph where the nodes represent states, and the edges signify transitions between states under the application of operators. This representation as a transition system can be used to derive heuristic functions, as we will later see in the case of abstraction heuristics. This section about transition systems and abstractions follows the explanations of Helmert et al. (2008).

Definition 3 (Transition system)**.** A *transition system* is a tuple $\mathcal{T} = \langle S, L, T, s_0, S_*, cost \rangle$ where

- S is a set of states where each state $s \in S$ describes a node in the graph,
- L is a set of labels,
- $T \subseteq S \times L \times S$ is a transition relation where every transition $t = \langle s, l, s' \rangle \in T$ describes a labelled edge $s \xrightarrow{l} s'$ in the graph,
- $s_0 \in S$ is the initial state,
- $S_* \subseteq S$ is a set of states that represent the goal states, and
- $\mathit{cost} : L \to \mathbb{N}_0$ is a cost function for every label $l \in L$.

A planning task $\Pi = \langle V, O, s_0, s_*, \mathit{cost} \rangle$ induces a transition system

$$\begin{aligned}
&\mathcal{T} = \langle S(\Pi), O, T, s_0, S_*, \mathit{cost} \rangle, \text{ where} \\
&T = \big\{ \langle s, o, s' \rangle \mid s, s' \in S(\Pi), o \in O, o \text{ is applicable in } s \text{ and } s[o] = s' \big\}, \text{ and} \\
&S_* = \{ s \in S(\Pi) \mid s_* \subseteq s \}.
\end{aligned}$$

An abstract task, or abstraction, is a transition system that represents a simplification of a planning task which ignores some aspects of the original task by merging states as defined by an abstraction function. Abstractions can be leveraged to create an admissible heuristic.

Definition 4 (abstraction)**.** Let Π be a planning task and let $\mathcal{T} = \langle S, L, T, s_0, S_*, \mathit{cost} \rangle$ be its transition system. The abstraction function $\alpha : S \to S^\alpha$ maps states to abstract states. The transition system $\mathcal{T}^\alpha = \langle S^\alpha, L, T', \alpha(s_0), S'_* \rangle$ where $T' = \{ \langle \alpha(s), l, \alpha(s') \rangle \mid \langle s, l, s' \rangle \in T \}$ and $S'_* = \{ \alpha(s_*) \mid s_* \in S_* \}$ is called the *induced abstraction* of $\mathcal{T}$ under α.

Projections are a concrete class of abstractions suitable for SAS$^+$ planning tasks. A projection includes only a subset of state variables of the original planning task:

Definition 5 (projection)**.** Let $\Pi = \langle V, O, s_0, s_*, \mathit{cost} \rangle$ be a planning task with a set of state variables V, and let $\mathcal{T}$ be the transition system of Π. The *projection* $\mathcal{T}^{V'}$ of $\mathcal{T}$ onto a *pattern*, that is a subset of variables $V' \subseteq V$, is the induced abstraction $\mathcal{T}^\alpha$ where $\alpha(s) = \alpha(s')$ if $s(v) = s'(v)$ for all $v \in V'$. An *atomic projection* $\mathcal{T}^{\{v\}}$ is a projection onto a single variable $v \in V$.

Every abstraction defines a heuristic function via the goal distance in its transition system:

Definition 6 (abstraction heuristic)**.** Let $\Pi = \langle V, O, s_0, s_*, \mathit{cost} \rangle$ be a planning task, let $\mathcal{T}$ be its transition system, and let $\mathcal{T}^\alpha = \langle S, L, T, s_0, S_*, \mathit{cost} \rangle$ be the induced abstraction of $\mathcal{T}$ under an abstraction function α. The *abstraction heuristic* h^α assigns every state $s \in S(\Pi)$ the cost of the shortest path in the abstraction $\mathcal{T}^\alpha$ from the abstract state $\alpha(s)$ to any goal state $\alpha(s_*) \in S_*$.

Since an abstraction is a homomorphism, i.e. transitions are being preserved, the abstraction heuristic h^α is admissible and consistent (Helmert et al., 2008).

A *pattern database* (Edelkamp, 2001) is a practical method for using patterns of a planning task as a basis for generating admissible heuristic estimates: in a precomputation step

before the actual search, all the transition systems of desired patterns are constructed. The shortest goal distances of all abstract states to their respective goal state are then stored in a data structure, usually some sort of hash table. During the search this data structure enables quick retrieval of the stored goal distances to serve as heuristic estimates for any state. A disadvantage of this method is that the memory requirements for storing all goal distances can be infeasible for larger patterns, which are often the most informative.

2.4 Operator-Counting

In recent years several heuristics based on linear programming have proven to be practical in optimal planning. Without going into details of the very diverse kinds of linear programming heuristics, we will summarize the *operator-counting framework* (Pommerening et al., 2014). The operator-counting framework is a framework for heuristics that are based on linear programming and it specifies a predefined structure for the constraints and objective function of said heuristics. The operator-counting framework allows different linear programming heuristics, provided that they can be expressed as so-called *operator-counting constraints*, to be compared on the basis of their constraints.

Definition 7 (operator-counting constraints)**.** Let $\Pi = \langle V, O, s_0, s_*, cost \rangle$ be a planning task and let $s \in S(\Pi)$ be a state. The integer variables $Count_o$ for each operator $o \in O$ are called *operator-counting variables*. Let π be an s-plan and let $occur(o, \pi)$ denote the number of times that operator o is applied in π. A linear inequality c over operator-counting variables is called an *operator-counting constraint* for state s if for every s-plan π, there exists a feasible solution to c with $Count_o = occur(o, \pi)$ for all $o \in O$.

Operator-counting constraints serve as constraints for the operator-counting integer/linear program.

Definition 8 (operator-counting integer/linear program)**.** Let $\Pi = \langle V, O, s_0, s_*, cost \rangle$ be a planning task. The *operator-counting integer program* IP_C for the set of operator-counting constraints C is

$$\text{minimize} \sum_{o \in O} cost(o) \cdot Count_o \text{ subject to } C$$

Given a state $s \in S(\Pi)$ and constraint set C for s, the LP heuristic $h^{LP}_C(s)$ is the objective value of the linear program $\mathrm{LP}_C(s)$, which is the LP relaxation of $\mathrm{IP}_C(s)$. The heuristic value is ∞ if the LP is infeasible.

As noted by Korte and Vygen (2006), there exist polynomial-time algorithms for solving Linear Programs, yet there are no known algorithms that are able to solve Integer Programs in polynomial time. In consequence, it is often useful to lift the integer restrictions on the operator-counting variables when computing IP/LP heuristics.

Intuitively, operator-counting constraints convey some information about how often an operator is applied in an optimal plan. Pommerening et al. (2014) provide several specific examples where the information can be derived from: they show that disjunctive action landmarks (Zhu and Givan, 2003), the state-equation heuristic (Bonet, 2013), optimal cost

partitioning (Katz and Domshlak, 2010), as well as the post-hoc optimization heuristic (Pommerening et al., 2013) can all be expressed as operator-counting constraints. Pommerening et al. (2014) also note that adding operator-counting constraints to an existing constraint set can never result in decreased heuristic estimates of the operator-counting linear program since an additional constraint only ever reduces the set of feasible solutions.

2.4.1 Post-Hoc Optimization Constraints

Post-hoc optimization constraints (Pommerening et al., 2013) are a specific type of operator-counting constraint that set operator-counting variables in relation to an admissible heuristic. Post-hoc optimization constraints "ignore" non-contributing operators, which are operators that do not have an influence on the corresponding heuristic.

Definition 9 (non-contributing operators)**.** Let $\Pi = \langle V, O, s_0, s_*, \mathit{cost}\rangle$ be a planning task, let $s \in S(\Pi)$ be a state, and let h be a heuristic that is admissible under cost. Let cost' be a cost function such that the heuristic value $h(s, \mathit{cost})$ remains an admissible estimate for state s in the planning task $\Pi' = \langle V, O, s_0, s_*, \mathit{cost}'\rangle$ with the cost function

$$\mathit{cost}'(o) = \begin{cases} 0 & \text{if } o \in N \\ \mathit{cost}(o) & \text{otherwise.} \end{cases}$$

Expressed formally, we require that $h(s, \mathit{cost}) \leq h^*(s, \mathit{cost}')$. Then, the set $N \subseteq O$ is called a set of *non-contributing operators.*

A Post-hoc optimization constraint is then an inequality over the remaining contributing operators in relation to the corresponding heuristic value:

Definition 10 (post-hoc optimization constraint)**.** Let $\Pi = \langle V, O, s_0, s_*, \mathit{cost}\rangle$ be a planning task, let $s \in S(\Pi)$ be a state, let h be an admissible heuristic under cost, and let N be a set of non-contributing operators. The post-hoc optimization constraint $c^{\mathrm{PhO}}_{h,N,\mathit{cost},s}$ is the constraint

$$c^{\mathrm{PhO}}_{h,N,\mathit{cost},s} : \sum_{o \in O \setminus N} \mathit{cost}(o) \cdot \mathit{Count}_o \geq h(s, \mathit{cost}).$$

Pommerening et al. (2014) provide a representation of pattern database heuristics as post-hoc operators constraints: operators that do not *affect* a given projection $\mathcal{T}^\alpha$, that is, they do not change a variable in the abstraction and thus do not induce any state-changing transition, do not influence the abstraction heuristic h^α. In consequence, these operators can be seen as non-contributing and the post-hoc optimization constraint of a pattern database heuristic is defined as follows:

Definition 11 (pattern database constraint)**.** Let $\Pi = \langle V, O, s_0, s_*, \mathit{cost}\rangle$ be a planning task and let $\mathcal{T}^\alpha$ be a projection of Π. The pattern database constraint $c^{\mathrm{PDB}}_{\alpha,\mathit{cost},s}$ is the constraint

$$c^{\mathrm{PDB}}_{\alpha,\mathit{cost},s} : \sum_{\substack{o \in O, \\ o \text{ affects } \mathcal{T}^\alpha}} \mathit{cost}(o) \cdot \mathit{Count}_o \geq h^\alpha(s, \mathit{cost}).$$

3

Cost-Altered Post-Hoc Optimization Constraints

Our contribution is to evaluate post-hoc optimization constraints under altered cost functions. A minimum solution of the operator-counting linear program under a set of regular post-hoc optimization constraints may be infeasible when the same type of constraints are evaluated under a new cost function. We propose that with this method of altering operator costs in post-hoc optimization constraints, we can obtain a more informed heuristic in some cases. In this section we provide a formal definition of such *cost-altered post-hoc optimization constraints* and we show that they are operator-counting constraints. We then examine the specific case of cost-altered pattern database constraints more thoroughly and we provide an example where the operator-counting heuristic derived from cost-altered pattern database constraints dominates regular pattern database constraints under the original cost function.

Definition 12 (cost-altered post-hoc heuristic constraint)**.** Let $\Pi = \langle V, O, s_0, s_*, \mathit{cost}\rangle$ be a planning task, let $s \in S(\Pi)$ be a state, let $h(s, \mathit{cost})$ be an admissible heuristic for Π under cost, and let N be a set of non-contributing operators. Let the cost function cost' be the cost function that we introduce. We define the *cost-altered post-hoc optimization heuristic constraint* $c^{\mathrm{PhO}}_{h,N,\mathit{cost}',s}$ as

$$c^{\mathrm{PhO}}_{h,N,\mathit{cost}',s} : \sum_{o \in O \setminus N} \mathit{cost}'(o) \cdot \mathit{Count}_o \geq h(s, \mathit{cost}').$$

We will now show that the cost-altered post-hoc optimization constraint $c^{\mathrm{PhO}}_{h,N,\mathit{cost}',s}$ is an operator-counting constraint.

Theorem 1. Let $\Pi = \langle V, O, s_0, s_*, \mathit{cost}\rangle$ be a planning task, let $s \in S(\Pi)$ be a state, let $h(s, \mathit{cost})$ be a heuristic that is admissible under cost, and let N be a set of non-contributing operators. Let the cost function cost' be the cost function that we introduce. The cost-altered post-hoc optimization heuristic constraint $c^{\mathrm{PhO}}_{h,N,\mathit{cost}',s}$ is an operator-counting constraint.

Proof. Let π be an s-plan for Π. We are going to show that the variable assignment $\mathit{Count}(o) = \mathit{occur}(o, \pi)$ corresponding to the s-plan π represents a feasible solution for

the constraint $c^{\text{PhO}}_{h,N,cost',s}$, i.e.

$$\sum_{o \in O \setminus N} cost'(o) \cdot occur(o, \pi) \geq h(s, cost').$$

Let $cost''$ be a cost function for Π defined as

$$cost''(o) = \begin{cases} 0 & \text{if } o \in N, \\ cost'(o) & \text{otherwise.} \end{cases}$$

It follows that

$$\begin{aligned}
&\sum_{o \in O \setminus N} cost'(o) \cdot occur(o, \pi) \\
&= \sum_{o \in O \setminus N} cost''(o) \cdot occur(o, \pi) && (cost''(o) = cost'(o) \text{ for all } o \in O \setminus N) \\
&= \sum_{o \in O} cost''(o) \cdot occur(o, \pi) && (cost''(o) = 0 \text{ for all } o \in N) \\
&= cost''(\pi) \\
&\geq h^*(s, cost'') && (h \text{ is admissible under } cost'') \\
&\geq h(s, cost') && (\text{Definition 9: } N \text{ is a set of non-contributing operators})
\end{aligned}$$

□

Note that it is not always the case that a heuristic h, while admissible under the cost function $cost$, is also admissible under every other arbitrary cost function $cost'$. Consider the heuristic

$$h(s) = \begin{cases} |\pi^*| & \text{if an optimal } s\text{-plan } \pi^* \text{ exists,} \\ \infty & \text{otherwise,} \end{cases}$$

which is an admissible heuristic for all planning tasks where $cost(o) \geq 1$ for all $o \in O$, but is not admissible in a planning task where $cost'(o) < 1$ for all $o \in O$.

3.1 Cost-Altered Pattern Database Constraints

For the remainder of this section, we will focus on the specific case of cost-altered pattern database constraints. We will show that they are operator-counting constraints under arbitrary cost functions and we will provide an example of a planning task where cost-altered pattern database constraints dominate regular pattern database constraints.

Corollary 1. Let Π be a planning task. The abstraction heuristic h^α is admissible under all cost functions, i.e.

$$h^\alpha(s, cost) \leq h^*(s, cost) \text{ for all cost functions } cost.$$

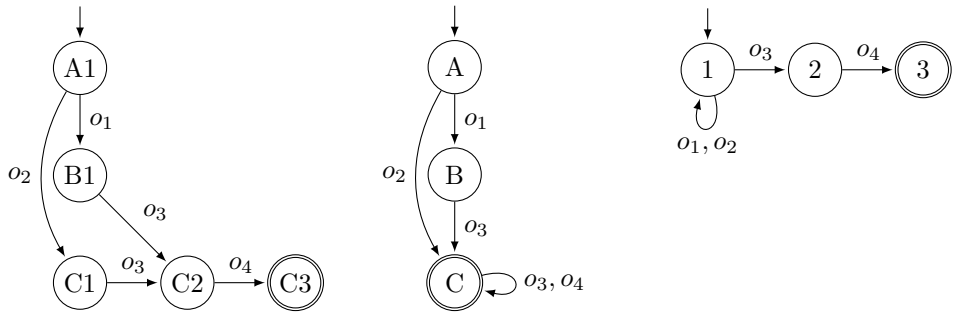
Proof. Follows directly from the fact that abstractions are transition-preserving and thus also plan-preserving, and the definition of abstraction heuristics in Definition 6. □

As a consequence of Corollary 1, we can freely choose the altered cost functions we want to introduce in pattern database constraints. We will call such pattern database constraints that are evaluated under an altered cost function *cost-altered pattern database constraints* and they are defined as follows:

Definition 13 (cost-altered pattern database constraint)**.** Let $\Pi = \langle V, O, s_0, s_*, cost \rangle$ be a planning task, let $s \in S(\Pi)$ be a state, let $V' \subseteq V$ be a pattern, let $\mathcal{T}^{V'}$ be the induced abstraction of Π under the pattern V', and let $h^{V'}$ be its abstraction heuristic. Let $cost'$ be the cost function we introduce. Then, the *cost-altered pattern database constraint* is defined as

$$c^{\text{PDB}}_{V',cost',s} : \sum_{\substack{o \in O, \\ o \text{ affects } \mathcal{T}^{V'}}} cost'(o) \cdot Count_o \geq h^{V'}(s, cost').$$

Having established that cost-altered post-hoc constraints are operator-counting constraints, we are now interested in the behavior of the operator-counting linear program under such constraints: in Figure 3.1 there is a planning task for which cost-altered pattern database constraints dominate pattern database constraints under the task's original cost function, as will now show in detail.



(a) Transition system $\mathcal{T}$ of a planning task Π.

(b) Transition system of the projection $\mathcal{T}^{\{a\}}$.

(c) Transition system of the projection $\mathcal{T}^{\{b\}}$.

Figure 3.1: A planning task where cost-altered post-hoc optimization constraints dominate regular post-hoc optimization constraints.

Let $\Pi = \langle \{a, b\}, \{o_1, o_2, o_3, o_4\}, s_0, s_*, \{o_1 \mapsto 7, o_2 \mapsto 10, o_3 \mapsto 7, o_4 \mapsto 6\} \rangle$ be the planning task shown in Figure 3.1a. Let $\mathcal{T}^{\{a\}}$ be its atomic projection under variable a, shown in Figure 3.1b. The atomic projection $\mathcal{T}^{\{b\}}$ is shown in Figure 3.1c. The variable domains are $dom(a) = \{A, B, C\}$, and $dom(b) = \{1, 2, 3\}$, respectively. The corresponding pattern database constraints $c_a = c^{\text{PDB}}_{\{a\},cost,s_0}$ of projection $\mathcal{T}^{\{a\}}$ and $c_b = c^{\text{PDB}}_{\{b\},cost,s_0}$ of projection $\mathcal{T}^{\{b\}}$ are

$$c_a : 7 \cdot Count_{o_1} + 10 \cdot Count_{o_2} + 7 \cdot Count_{o_3} + 6 \cdot Count_{o_4} \geq h^{\{a\}}(s, cost) = 10, \text{ and}$$
$$c_b : 7 \cdot Count_{o_1} + 10 \cdot Count_{o_2} + 7 \cdot Count_{o_3} + 6 \cdot Count_{o_4} \geq h^{\{b\}}(s, cost) = 13.$$

We now change the cost of operator o_3 from 7 to 4 and introduce our altered cost function : $cost' = \{o_1 \mapsto 7, o_2 \mapsto 10, o_3 \mapsto 4, o_4 \mapsto 6\}$. The cost-altered pattern database constraint

$c'_a = c^{\text{PDB}}_{\{a\},cost',s_0}$ of the projection $\mathcal{T}^{\{a\}}$ under $cost'$ is then defined as

$$c'_a : 7 \cdot Count_{o_1} + 10 \cdot Count_{o_2} + 4 \cdot Count_{o_3} + 6 \cdot Count_{o_4} \geq h^{\{a\}}(s, cost') = 10.$$

Let us now examine the behavior of the operator-counting linear program under these constraints. Let s_0 be the initial state of planning task Π. The optimal solution $Count^*$ to the operator-counting linear program $LP_{\{c_a,c_b\}}(s_o)$ subject to the regular post-hoc optimization constraints c_a and c_b for state s_0 is

$$Count^* = \operatorname*{argmin}_{Count} \Big\{ \sum_{o \in O} cost(o) \cdot Count_o \mid \{c_a, c_b\} \Big\} = \{o_1 \mapsto 0, o_2 \mapsto 0, o_3 \mapsto 2, o_4 \mapsto 0\},$$

which results in the heuristic estimate $h^{LP}_{\{c_a,c_b\}}(s_0, cost) = 14$.

Note that the solution $Count^*$ does not represent an actual plan in the planning task Π, and that it violates the cost-altered post-hoc optimization constraint c'_a. In consequence, the solution $Count^*$ will be infeasible in a linear program subject to a constraint set that contains c'_a. When we evaluate the operator-counting linear program $LP_{\{c'_a,c_b\}}(s_o)$ we arrive at a different minimum solution

$$Count^{*'} = \operatorname*{argmin}_{Count} \Big\{ \sum_{o \in O} cost(o) \cdot Count_o \mid \{c'_a, c_b\} \Big\} = \{o_1 \mapsto 1, o_2 \mapsto 0, o_3 \mapsto 1, o_4 \mapsto 1\}.$$

Indeed, the heuristic estimate $h^{LP}_{\{c'_a,c_b\}}(s_0, cost) = 20 > h^{LP}_{\{c_a,c_b\}}(s_0, cost) = 14$ for initial state s_0 has improved under the constraint set that includes the constraint c'_a, while the heuristic estimates are identical between both constraint sets for all other states in the planning task.

4

Experimental Results

To examine the behavior of cost-altered post-hoc optimization constraints in practice, we used the existing implementation of the operator-counting framework present in the Fast Downward planning system (Helmert, 2011) and extended it so that cost-altered post-hoc optimization constraints are supported. In this section we present the experiment setup as well as a discussion of benchmark results we obtained.

4.1 Experiment Setup

In our evaluations we used the A^* search algorithm to find a plan, in conjunction with the operator-counting heuristic h_C^{LP}, where C is a set of operator-counting constraints. In our experiments we evaluated the following sets of operator-counting constraints, as well as all combinations thereof.

SEQ includes all lower-bound net change constraints (Bonet, 2013, Pommerening et al., 2014),

LMC adds a landmark constraint for each disjunctive action landmark (Pommerening et al., 2014, Zhu and Givan, 2003),

PhO_Norm includes all pattern database constraints for systematically generated patterns up to a size of 2 variables[1] (Pommerening et al., 2014),

PhO_One includes all cost-altered pattern database constraints for systematically generated patterns up to a size of 2 variables with the cost function $cost(o) = 1$ for all operators,

PhO_Rand includes all cost-altered pattern database constraints for systematically generated patterns up to a size of 2 variables, where the altered cost function assigns each operator a random cost between 1 and its original cost.

[1] Pommerening et al. (2014) have determined this method of pattern generation to be the most successful in their experiments. We have not evaluated different pattern generation methods during the course of our experiments.

The LP solver used in the experiments is the IBM ILOG CPLEX Optimization Studio v12.8.0. All experiments were run on Intel Xeon E5-2690 processors with 2.60 GHz cores under a time limit of 30 minutes and a memory limit of 3584 MB per task.

The set of planning problem domains that we evaluated was chosen from a suite of planning tasks in several domains[2] that have been used in the International Planning Competitions of recent years. A specific subset of tasks were selected from this suite which are formulated in planning domains that

- are suitable for optimal planning,
- do note consist solely of unit-cost operators,
- do not contain axioms or conditional operators, as these are not supported by the operator-counting framework in Fast Downward.

4.2 Evaluation of Results

In this section we will evaluate our experiment results on the different constraint sets mentioned above. We begin by evaluating the constraints sets in isolation first and then in combination.

We will first focus on the constraint set PhO_One. See the left hand side of Table 4.1 for coverage results of our benchmarks on different types of operator-counting constraints in on their own. We find an overall reduced coverage of PhO_One in comparison to PhO_Norm. The constraint set PhO_One never improves over SEQ except where PhO_Norm also does. Still, some specific domains show interesting results: in the *scanalyzer* domain PhO_One has improved coverage over PhO_Norm. On the other hand, coverage is drastically reduced compared to PhO_Norm in the *elevators*, *spider*, and *woodworking* domains. An examination of the initial heuristic values has also confirmed that in comparison to PhO_Norm they have slightly improved in the *scanalyzer* and *tetris* domains only, and worsened otherwise.

PhO_Rand has achieved an overall significantly lower coverage than the other evaluated constraint sets. It seems that this method of handing out operator costs is too simplistic and removes information rather than cause any heuristic improvement. From our experiments we conclude that, in order to consistently achieve improved heuristic estimates, a more informed and sophisticated method for finding desirable cost functions would be necessary. On the other hand, it seems unclear where the information for generating such a cost function would come from if not from domain-dependent analysis. Another open question is whether this cost function could be generated efficiently enough to be practical.

For combinations of cost-altered pattern database constraints together with other types of operator-counting constraints (see right side of Table 4.1), we have found that we achieved overall slightly reduced or similar coverage to constraint sets without cost-altered constraints. This is in line with our observation that, while a cost-altered pattern database heuristic

[2] The set was retrieved from https://bitbucket.org/aibasel/downward-benchmarks.

can be computed efficiently, there is generally no heuristic information gained by the cost-altering.

Coverage	**PhO_Norm**	**PhO_One**	**PhO_Rand**	**LMC**	**SEQ**	**LMC+ PhO_One**	**LMC+ PhO_Rand**	**LMC+ PhO_Norm**	**SEQ+ LMC**
agricola-opt18-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
barman-opt11-strips (20)	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
data-network-opt18-strips (20)	9	7	8	**12**	6	**12**	**12**	**12**	**12**
elevators-opt08-strips (30)	18	9	10	**20**	9	19	19	**20**	19
elevators-opt11-strips (20)	15	7	8	**16**	7	**16**	**16**	**16**	**16**
floortile-opt14-strips (20)	0	0	0	**5**	2	**5**	**5**	3	**5**
ged-opt14-strips (20)	**15**	13	13	**15**	13	13	**15**	**15**	10
nomystery-opt11-strips (20)	**16**	**16**	8	14	10	**16**	14	**16**	12
openstacks-agl14-strips (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
openstacks-opt08-strips (30)	**17**	15	**17**	**17**	16	15	**17**	**17**	16
openstacks-opt11-strips (20)	**12**	10	**12**	**12**	11	11	**12**	**12**	11
openstacks-opt14-strips (20)	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
organic-synthesis-split-opt18-strips (20)	7	7	7	15	10	13	15	13	**16**
parcprinter-08-strips (30)	17	15	12	18	28	18	18	19	**29**
parcprinter-opt11-strips (20)	13	11	9	13	**20**	13	13	14	**20**
parking-opt11-strips (20)	1	1	0	2	**3**	1	1	1	2
parking-opt14-strips (20)	2	1	0	**3**	**3**	0	2	0	**3**
pegsol-08-strips (30)	27	25	27	27	**28**	27	27	27	**28**
pegsol-opt11-strips (20)	17	15	17	17	**18**	17	17	17	**18**
petri-net-alignment-opt18-strips (20)	2	1	2	9	8	8	9	9	**10**
scanalyzer-08-strips (30)	7	14	6	**15**	14	14	13	13	13
scanalyzer-opt11-strips (20)	4	11	3	**12**	11	11	10	10	10
sokoban-opt08-strips (30)	**28**	**28**	24	**28**	18	**28**	**28**	**28**	**28**
sokoban-opt11-strips (20)	**20**	**20**	**20**	**20**	15	**20**	**20**	**20**	**20**
spider-opt18-strips (20)	11	6	6	11	**13**	6	6	9	11
tetris-opt14-strips (17)	3	3	2	5	**12**	3	3	2	11
transport-opt08-strips (30)	**11**	**11**	10	**11**	10	**11**	**11**	**11**	**11**
transport-opt11-strips (20)	**6**	**6**	5	**6**	5	**6**	**6**	**6**	**6**
transport-opt14-strips (20)	4	4	4	**6**	4	**6**	**6**	**6**	**6**
woodworking-opt08-strips (30)	15	10	7	16	13	16	16	18	**21**
woodworking-opt11-strips (20)	10	5	2	11	8	11	11	13	**16**
Sum (697)	312	276	244	361	320	341	347	352	**385**

Table 4.1: Coverage results for the different operator-counting constraint sets we evaluated and of a selection of combinations of them.

5
Conclusion

In this thesis, we have introduced cost-altered post-hoc optimization constraints and we have shown that they are operator-counting constraints. Furthermore, We have shown that it is possible that the operator-counting linear heuristic can achieve a higher heuristic estimate under cost-altered post-hoc optimization constraints in comparison with post-hoc optimization constraints under the original cost function.

In our experiments we found that we achieved generally lower problem coverage in comparison to regular post-hoc optimization constraints. Still, cost-alteration has improved coverage and initial heuristic values in a small number of problem domains even though the methods for choosing new cost functions that we evaluated are quite simplistic. More structured methods for finding alternate cost functions may lead to better results, although finding a cost function with desirable properties is probably also somewhat domain-dependent and may be computationally expensive. Furthermore, we have only evaluated cost-alteration on post-hoc optimization constraints for pattern databases. The heuristics of pattern databases are guaranteed to be admissible under all cost functions and the heuristic values under the altered cost function can efficiently be retrieved. For other heuristics, both these handy properties may not be true. Still, cost-altered post-hoc optimization constraints may be an appropriate heuristic technique when structural properties of the task allow a desirable cost function to be found.

Bibliography

Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–656, 1995.

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.

Stefan Edelkamp. Planning with pattern databases. In *Proceedings of the Sixth European Conference on Planning*, pages 84–90, 2001.

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 AAAI Conference on Artificial Intelligence*, pages 1007–1012, 2007.

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2011.

Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Explicit-state abstraction: A new method for generating heuristic functions. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1547–1550, 2008.

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

Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Springer, 2006.

Florian Pommerening, Gabriele Röger, and Malte Helmert. Getting the most out of pattern databases for classical planning. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 2357–2364, 2013.

Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. LP-based heuristics for cost-optimal planning. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, pages 226–234, 2014.

Stuart J. Russell and Peter Norvig. *Artificial intelligence - a modern approach*. Prentice Hall, 1995.

Lin Zhu and Robert Givan. Landmark extraction via planning graph propagation. In *Printed Notes of International Conference on Automated Planning and Scheduling 2003 Doctoral Consortium*, 2003.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

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

Author — Autor

Andreas Thüring

Matriculation number — Matrikelnummer

09-724-162

Title of work — Titel der Arbeit

Evaluation Of Post-Hoc Optimization Constraints Under Altered Cost Functions

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 6, 2019

Signature — Unterschrift