Saturated Post-hoc Optimization for Classical Planning

Jendrik Seipp,[1,2] **Thomas Keller,**[1] **Malte Helmert**[1]
[1]University of Basel, Switzerland
[2]Linköping University, Sweden
jendrik.seipp@liu.se, tho.keller@unibas.ch, malte.helmert@unibas.ch

Abstract

Saturated cost partitioning and post-hoc optimization are two powerful cost partitioning algorithms for optimal classical planning. The main idea of saturated cost partitioning is to give each considered heuristic only the fraction of remaining operator costs that it needs to prove its estimates. We show how to apply this idea to post-hoc optimization and obtain a heuristic that dominates the original both in theory and on the IPC benchmarks.

Introduction

Methods for admissibly combining admissible heuristics significantly improved the state of the art in optimal classical planning. Early work that exploits the additivity of heuristics (Felner, Korf, and Hanan 2004; Edelkamp 2006) led to the development of the *canonical heuristic* (Haslum et al. 2007), which computes all maximal subsets of pairwise independent *pattern database* (PDB) heuristics and then uses the maximum of all sums over these subsets as the heuristic value.

The additivity criterion has been replaced by a more general theory with the introduction of *cost partitioning* (Katz and Domshlak 2008), which ensures that admissible heuristic estimates can be summed up admissibly as long as the sum of the operator costs that are used to compute the heuristics does not exceed the original cost function. The computation of *optimal cost partitionings* (Katz and Domshlak 2010) has — despite its polynomial time complexity — proven to be too costly in practice (Pommerening, Röger, and Helmert 2013; Seipp, Keller, and Helmert 2017b). Suboptimal cost partitionings that are efficiently computable both in theory and practice have therefore received much attention.

The underlying observation of *saturated cost partitioning* (Seipp and Helmert 2014, 2018) is that we can often reduce operator costs without affecting heuristic estimates. The saturated cost partitioning algorithm considers a set of heuristics in a given order. For the first heuristic h, it computes all heuristic values of h under the original cost function, computes the minimum cost function that preserves the heuristic values, that is, *the saturated cost function*, and continues with the next heuristic with the remaining costs until all heuristics have been considered.

The *post-hoc optimization* heuristic (Pommerening, Röger, and Helmert 2013) uses a linear program (LP) to determine a real-valued factor for each heuristic in a set of *pattern database* (PDB) abstractions (Edelkamp 2001), and the cost partitioning is derived by multiplying the costs of all operators that affect an abstraction with that factor. Operators that only induce self-loops in an abstraction are assigned costs of zero. In the (equivalent) operator counting version of the dual LP (Pommerening et al. 2014), there is a constraint for each PDB that specifies that the total contribution of all operators in a heuristic multiplied with the operator cost must at least sum up to the heuristic estimate. The fact that every plan for the original problem must also be a plan for each PDB allows us to use the same operator counting variables for all abstractions.

In this paper, we strengthen the post-hoc optimization constraints by combining them with the central idea of saturated cost partitioning. We observe that an operator never contributes more than its saturated cost to the heuristic value of an abstraction because all heuristic values remain the same even if we were to reduce its cost to the possibly lower saturated cost. We show that the resulting *saturated post-hoc optimization* heuristic dominates post-hoc optimization theoretically and confirm that the theoretical superiority carries over into practice in an experimental evaluation on the domains of the International Planning Competitions.

Background

A classical planning task induces a transition system and solving the task optimally implies finding a shortest path from the initial state to a goal state. Formally, a *transition system* $\mathcal{T}$ is a directed, labeled graph defined by a finite set of *states* $S(\mathcal{T})$, a finite set of *labels* $L(\mathcal{T})$, a set $T(\mathcal{T})$ of labeled *transitions* $s \xrightarrow{\ell} s'$ with $s, s' \in S(\mathcal{T})$ and $\ell \in L(\mathcal{T})$, an *initial state* $s_0(\mathcal{T})$, and a set of *goal states* $S_\star(\mathcal{T})$.

A *goal path* from $s \in S(\mathcal{T})$ is a sequence of transitions from s to any goal state $s' \in S_\star(\mathcal{T})$. Goal paths are also called *plans*.

A *cost function* for transition system $\mathcal{T}$ is a function $cost : L(\mathcal{T}) \to \mathbb{R}$. A cost function *cost* is *non-negative* if $cost(\ell) \geq 0$ for all labels ℓ. We write $\mathcal{C}(\mathcal{T})$ for the set of all cost functions for $\mathcal{T}$. Our input planning tasks use non-negative cost functions, but following Pommerening et al.

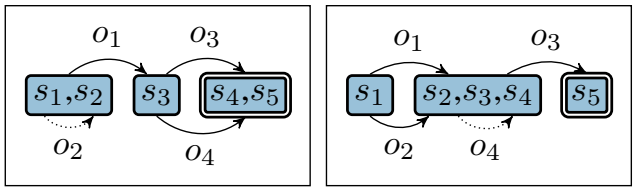

Figure 1: Example abstraction heuristics h_1 (left) and h_2 (right) from Seipp, Keller, and Helmert (2017a). The cost function is $\mathit{cost} = \{o_1 \mapsto 4, o_2 \mapsto 1, o_3 \mapsto 4, o_4 \mapsto 1\}$. The s_i are concrete states, the rounded rectangles are abstract states, abstract goal states use double borders and dotted lines depict abstract self-loops.

(2015) we allow negative costs in cost partitionings.[1]

By combining a transition system $\mathcal{T}$ and a cost function *cost*, we obtain a *weighted* transition system $\langle \mathcal{T}, \mathit{cost} \rangle$. The cost of a goal path in a weighted transition system is the sum of its transition weights and a goal path is *optimal* if its cost is minimal among all goal paths. The *goal distance* $h^*_{\mathcal{T}}(\mathit{cost}, s)$ of a state s in $\mathcal{T}$ is the cost of an optimal goal path from s under the cost function *cost* or ∞ if there is no goal path from s.

A *heuristic* is a function $h : \mathcal{C}(\mathcal{T}) \times S(\mathcal{T}) \to \mathbb{R} \cup \{-\infty, \infty\}$ that estimates the distance of a given state to a goal state under a given cost function (Pearl 1984). It is *admissible* if $h(\mathit{cost}, s) \leq h^*_{\mathcal{T}}(\mathit{cost}, s)$ for all cost functions $\mathit{cost} \in \mathcal{C}(\mathcal{T})$ and all states $s \in S(\mathcal{T})$. A heuristic h is *cost-monotonic* if $h(\mathit{cost}, s) \geq h(\mathit{cost}', s)$ for all states $s \in S(\mathcal{T})$ whenever $\mathit{cost} \geq \mathit{cost}'$ (that is, making transitions more expensive cannot decrease heuristic estimates).

In this paper, we focus on *abstraction heuristics* (e.g., Edelkamp 2001; Helmert, Haslum, and Hoffmann 2007; Katz and Domshlak 2008, 2010). An abstraction heuristic for a transition system $\mathcal{T}$ is defined by a transition system $\mathcal{T}'$ called the *abstract transition system* and a function $\alpha : S(\mathcal{T}) \to S(\mathcal{T}')$ called the *abstraction function*. The abstraction function must preserve goal states and transitions; we refer to the literature for details (Helmert, Haslum, and Hoffmann 2007). Heuristic values are computed by mapping states of $\mathcal{T}$ (*concrete states*) to states of $\mathcal{T}'$ (*abstract states*) and computing the goal distance in the abstract transition system: $h(\mathit{cost}, s) = h^*_{\mathcal{T}'}(\mathit{cost}, \alpha(s))$. Abstraction heuristics are cost-monotonic.

Cost partitioning is the preferable mechanism for combining multiple admissible heuristics. It preserves admissibility by distributing the costs of a task among the component heuristics (Katz and Domshlak 2008, 2010). A *cost partitioning* for a transition system $\mathcal{T}$ and a cost function $\mathit{cost} \in \mathcal{C}(\mathcal{T})$ is a tuple $\langle \mathit{cost}_1, \ldots, \mathit{cost}_n \rangle \in \mathcal{C}(\mathcal{T})^n$ whose sum is bounded by *cost*: $\sum_{i=1}^n \mathit{cost}_i(\ell) \leq \mathit{cost}(\ell)$ for all $\ell \in L(\mathcal{T})$. It induces the *cost-partitioned heuristic* $h(\mathit{cost}, s) = \sum_{i=1}^n h_i(\mathit{cost}_i, s)$.

[1]To cleanly state some definitions concerning saturated cost partitioning, it is useful to allow costs of positive and negative infinity. However, to simplify the presentation here, we only consider finite cost functions and refer to Seipp, Keller, and Helmert (2020) for details on the more general case.

Saturated Cost Partitioning

One of the strongest algorithms for computing cost partitionings is saturated cost partitioning (Seipp and Helmert 2014; Seipp, Keller, and Helmert 2020). At the heart of the algorithm lies the concept of *saturated cost functions*, which capture the insight that heuristic estimates often remain the same even if we evaluate the heuristic under a reduced cost function.

Definition 1 (saturated cost function)**.** *Consider a transition system $\mathcal{T}$, a heuristic h for $\mathcal{T}$ and a cost function $\mathit{cost} \in \mathcal{C}(\mathcal{T})$. A cost function $\mathit{scf} \in \mathcal{C}(\mathcal{T})$ is* saturated *for h and cost if*

1. *$\mathit{scf}(\ell) \leq \mathit{cost}(\ell)$ for all labels $\ell \in L(\mathcal{T})$ and*
2. *$h(\mathit{scf}, s) = h(\mathit{cost}, s)$ for all states $s \in S(\mathcal{T})$.*

We can compute a saturated cost function for any given heuristic h and cost function *cost*, since *cost* itself satisfies the criteria of Definition 1. Ideally, however, we want to find a saturated cost function that uses lower costs than the original cost function (to save costs for other heuristics). For some classes of heuristics, there is a unique *minimum* saturated cost function and we can compute it efficiently (Seipp, Keller, and Helmert 2017a, 2020).[2] This is the case for explicitly represented abstraction heuristics (Helmert et al. 2014) such as the pattern database heuristics (Haslum et al. 2007; Pommerening, Röger, and Helmert 2013) and Cartesian abstractions (Seipp and Helmert 2013) we consider below, and landmark heuristics (Karpas and Domshlak 2009).

Example 1. *Consider the abstraction heuristic h_1 in Figure 1. Under the original cost function cost, h_1 yields the following estimates: $h_1(s_1) = h_1(s_2) = 5$, $h_1(s_3) = 1$, $h_1(s_4) = h_1(s_5) = 0$. Reducing the costs to the minimum saturated cost function $\mathit{scf} = \{o_1 \mapsto 4, o_2 \mapsto 0, o_3 \mapsto 1, o_4 \mapsto 1\}$ does not change any heuristic estimates.*

Saturated cost partitioning iteratively computes saturated cost functions for a sequence of heuristics.

Definition 2 (saturated cost partitioning)**.** *Consider a weighted transition system $\langle \mathcal{T}, \mathit{cost} \rangle$ and an ordered sequence of heuristics $\omega = \langle h_1, \ldots, h_n \rangle$ for $\mathcal{T}$.*

The saturated cost partitioning $\langle \mathit{cost}_1, \ldots, \mathit{cost}_n \rangle$ *of the cost function cost for ω is defined as:*

$$
\begin{aligned}
&\mathit{rem}_0 = \mathit{cost} \\
&\mathit{cost}_i = \mathit{saturate}(h_i, \mathit{rem}_{i-1}) && \text{for all } 1 \leq i \leq n \\
&\mathit{rem}_i = \mathit{rem}_{i-1} - \mathit{cost}_i && \text{for all } 1 \leq i \leq n,
\end{aligned}
$$

where the auxiliary cost functions rem_i represent the remaining costs after processing the first i heuristics in ω and $\mathit{saturate}(h_i, \mathit{rem}_{i-1})$ computes a cost function that is saturated for h_i and rem_{i-1}. In this work, saturate always refers to the unique minimum saturated cost function for a given abstraction heuristic and cost function.

We write $h^{\mathrm{SCP}}_{\omega}$ for the heuristic that is cost-partitioned by saturated cost partitioning for order ω.

[2]Strictly speaking, this is only true if we allow costs of $-\infty$ to account for labels that a heuristic detects as "dead", i.e., not part of any goal path. We again refer to Seipp, Keller, and Helmert (2020) for details. Later on, we discuss a way to deal with such dead labels outside the cost partitioning mechanism.

Example 2. *Consider the two abstraction heuristics h_1 and h_2 in Figure 1. If we compute a saturated cost partitioning for the order $\langle h_1, h_2 \rangle$, we obtain the saturated cost function $cost_1 = \{o_1 \mapsto 4, o_2 \mapsto 0, o_3 \mapsto 1, o_4 \mapsto 1\}$ as in Example 1. Thus, we have remaining costs $rem_1 = \{o_1 \mapsto 0, o_2 \mapsto 1, o_3 \mapsto 3, o_4 \mapsto 0\}$. Under this cost function, we have $h_2(rem_1, s_i) = 3$ for $1 \leq i \leq 4$ and $h_2(rem_1, s_5) = 0$. Computing the saturated cost function for h_2 under rem_1 reveals that we can reduce $rem_1(o_2) = 1$ to $cost_2(o_2) = 0$, and could use its cost for hypothetical subsequent heuristics. The functions $cost_1$ and $cost_2$ form a cost partitioning, which means that we can sum their estimates admissibly to obtain $h^{\text{SCP}}_{\langle h_1, h_2 \rangle}(s_1) = h_1(cost_1, s_1) + h_2(cost_2, s_1) = 5 + 3 = 8$.*

Post-hoc Optimization

The second cost partitioning algorithm we consider in this paper is *post-hoc optimization* (Pommerening, Röger, and Helmert 2013). The post-hoc optimization heuristic h^{PhO} dominates the canonical heuristic h^{CAN} (Haslum et al. 2007), even though computing h^{PhO} takes polynomial time, whereas computing h^{CAN} is NP-equivalent. Originally, post-hoc optimization has been applied to pattern database heuristics, but it has also been used for Cartesian abstractions and landmark heuristics since (Seipp, Keller, and Helmert 2017a). Like optimal cost partitioning, post-hoc optimization is based on linear programming (LP), but since the post-hoc optimization LPs are much smaller than LPs for optimal cost partitioning over the same set of heuristics, the former are much faster to solve in practice. Consequently, in contrast to optimal cost partitioning, it is possible to compute a cost partitioning based on post-hoc optimization for each evaluated state during an A* search and obtain competitive performance.

A central concept for post-hoc optimization is the distinction whether or not a given label *affects* a given heuristic. We follow the definition by Seipp, Keller, and Helmert (2017a):

Definition 3 (affects)**.** *A label ℓ* affects *a heuristic h for a transition system $\mathcal{T}$ if there exists a state $s \in S(\mathcal{T})$ and two non-negative cost functions cost and cost$'$ differing only on ℓ with $h(cost, s) \neq h(cost', s)$.*

Note the restriction to non-negative cost functions in this definition. If negative cost functions were considered, the definition would trivialize in many cases, failing to capture the traditional notion of labels affecting a heuristic in the literature on which post-hoc optimization is based. In particular, every label with negative cost that induces a self-looping transition along any goal path in an abstract transition system would force the abstraction heuristic value to $-\infty$, and hence all these self-loop-inducing labels would have to be considered as affecting the heuristic. This is not in the spirit of the post-hoc optimization approach and would lead to inferior heuristic estimates.

Note also that the definition is purely semantic: it is based on heuristic values under different cost functions, without regard to how the heuristic is computed or how the property can be evaluated. To make computations based on affecting labels practically efficient, this purely semantic notion is generally relaxed. In the following, we write $\mathcal{A}(h)$ to denote an overapproximation of the set of all labels that affect the heuristic h. For abstraction heuristics h, which are the only heuristics we experimentally evaluate in this paper, we set $\mathcal{A}(h)$ to the set of all labels ℓ such that there exists a transition $s \xrightarrow{\ell} s'$ with $s \neq s'$ in the abstract transition system. This set is guaranteed to include all labels that affect the heuristic, but may also include some labels that do not affect the heuristic. The post-hoc optimization heuristic is admissible with any overapproximation of affecting labels, but tighter sets can improve the quality of the heuristic.

We can now describe the linear program defining the post-hoc optimization heuristic. All linear programs can be written in *primal* and *dual* form. We call the following LP the primal post-hoc optimization LP and refer to it as PhO-LP:

Definition 4 (h^{PhO})**.** *Let $\langle \mathcal{T}, cost \rangle$ be a weighted transition system with labels L, and let $\mathcal{H}$ be a set of admissible heuristics for $\mathcal{T}$. The* post-hoc optimization heuristic $h^{\text{PhO}}(cost, s)$ *for state $s \in S(\mathcal{T})$, $\mathcal{T}$ and $\mathcal{H}$ is the objective value of the PhO-LP:*

$$\text{minimize} \sum_{\ell \in L} cost(\ell) \cdot Y_\ell \text{ s.t.}$$

$$\sum_{\ell \in \mathcal{A}(h)} cost(\ell) \cdot Y_\ell \geq h(cost, s) \text{ for all } h \in \mathcal{H} \quad (1)$$

$$Y_\ell \geq 0 \text{ for all } \ell \in L \quad (2)$$

The PhO-LP uses the operator counting formulation of post-hoc optimization (Pommerening et al. 2014). We can transform this formulation into the equivalent original formulation by Pommerening, Röger, and Helmert (2013) if we replace $cost(\ell) \cdot Y_\ell$ by X_o, where Y_ℓ counts the number of times label ℓ is used and X_o stands for the induced costs by operator o.

By converting the PhO-LP into its dual formulation PhO-Dual, we can see the relationship to cost partitioning more clearly:

$$\text{maximize} \sum_{h \in \mathcal{H}} h(cost, s) \cdot w_h \text{ s.t.}$$

$$\sum_{h \in \mathcal{H}: \ell \in \mathcal{A}(h)} cost(\ell) \cdot w_h \leq cost(\ell) \text{ for all } \ell \in L \quad (3)$$

$$w_h \geq 0 \text{ for all } h \in \mathcal{H}. \quad (4)$$

In the resulting post-hoc optimization cost partitioning, each heuristic $h \in \mathcal{H}$ is assigned the cost function $cost_h$, where $cost_h(\ell) = cost(\ell) \cdot w_h$ if $\ell \in \mathcal{A}(h)$ and $cost_h(\ell) = 0$ otherwise. Due to the PhO-Dual constraints, all such cost functions $cost_h$ are non-negative.[3]

[3]Note that we could simplify Constraint 3 to the inequality $\sum_{h \in \mathcal{H}: \ell \in \mathcal{A}(h)} w_h \leq 1$ by dividing both sides by $cost(\ell)$ and omitting the constraint for zero-cost labels, for which it is trivially satisfied. Intuitively, if a label affects multiple heuristics, then the total weight we assign to these heuristics cannot exceed 1. This ensures that we do not account for a higher cost contributed by the label than it actually contributes.

Saturated Post-hoc Optimization

The main observation we exploit in this paper is that it is impossible that an operator contributes more than its saturated cost to the heuristic value of an abstraction, which implies that we can use the saturated cost of an operator in Constraint 1 rather than the (original) cost of the operator.

Definition 5 (h^{SPhO})**.** *Let $\langle \mathcal{T}, cost \rangle$ be a weighted transition system with labels L, and let $\mathcal{H}$ be a set of admissible heuristics for $\mathcal{T}$. The* saturated post-hoc optimization heuristic $h^{\text{SPhO}}(cost, s)$ *for state $s \in S(\mathcal{T})$, $\mathcal{T}$ and $\mathcal{H}$ is the objective value of the SPhO-LP:*

$$\text{minimize} \sum_{\ell \in L} cost(\ell) \cdot Y_\ell \text{ s.t.}$$

$$\sum_{\ell \in L} scf_h(\ell) \cdot Y_\ell \geq h(cost, s) \text{ for all } h \in \mathcal{H} \tag{5}$$

$$Y_\ell \geq 0 \text{ for all } \ell \in L, \tag{6}$$

where scf_h is a saturated cost function for h and cost.

Of course, different choices of scf_h lead to different heuristics. In the following, we focus on heuristics h for which a unique minimum saturated cost function exists and the case where scf_h is this function. As discussed earlier, this is the case for abstraction heuristics and more generally all heuristics to which the concepts of post-hoc optimization or saturated cost functions have been applied in the literature.

In cases where a heuristic function detects that a label ℓ cannot be part of any goal path, $scf_h(\ell)$ could be set arbitrarily low (to $-\infty$ if we considered infinite cost functions). Because linear programs require finite coefficients, we instead add the constraint $Y_\ell = 0$ in such cases and omit terms for Y_ℓ in Constraint 5, which is mathematically equivalent to dealing with such infinities directly.

There are two notable differences between the SPhO-LP and the PhO-LP (and indeed these are the only differences). Firstly, Constraint 5 uses the saturated cost function scf_h, while Constraint 1 uses the original cost function *cost*. Secondly, the SPhO-LP does not require a notion of affecting labels: the sum in Constraint 5 is over all labels, where Constraint 1 is restricted to (possibly an overapproximation of) the labels affecting the given heuristic. Indeed, this restriction is critical for the utility of h^{PhO}: it is not difficult to see from the dual formulation that, if we replaced $\mathcal{A}(h)$ with the set of *all* labels, h^{PhO} degenerates to the maximum of the component heuristics. Saturated post-hoc optimization can work without knowledge of which labels affect which heuristics because the same (and more) information is already implicit in the saturated cost function, as we will see in the following.

Before comparing h^{SPhO} to h^{PhO}, we first establish that h^{SPhO} is admissible.

Theorem 1. *h^{SPhO} is admissible. This result holds for all possible choices for the saturated cost functions scf_h.*

Proof. Consider SPhO-Dual, the dual of the SPhO-LP:

$$\text{maximize} \sum_{h \in \mathcal{H}} h(cost, s) \cdot w_h \text{ s.t.}$$

$$\sum_{h \in \mathcal{H}} scf_h(\ell) \cdot w_h \leq cost(\ell) \text{ for all } \ell \in L \tag{7}$$

$$w_h \geq 0 \text{ for all } h \in \mathcal{H}. \tag{8}$$

By the duality theorem (e.g., Schrijver 1998), the objective value of the SPhO-Dual is equal to the objective value of the SPhO-LP, so it is sufficient to show that the objective value of SPhO-Dual is an admissible estimate.

Consider any solution for SPhO-Dual. Because $h(scf_h, s)$ is an admissible heuristic value under the cost function scf_h and because $w_h \geq 0$, $h(scf_h, s) \cdot w_h$ is an admissible heuristic value under the cost function $scf_h \cdot w_h$. Constraint 7 is exactly the cost partitioning constraint under these (scaled by w_h) heuristics, and the objective value of the LP is exactly the cost-partitioned heuristic value, taking into account that $h(cost, s) = h(scf_h, s)$ by the properties of saturated cost functions. Admissibility thus follows from the admissibility of cost partitioning (Katz and Domshlak 2010; Pommerening et al. 2015). □

The advantage of saturated post-hoc optimization over regular post-hoc optimization is that the saturated cost of an operator can be significantly smaller than its cost. It can even be negative. This means that Constraint 5 can be significantly tighter than Constraint 1, leading to a higher admissible heuristic estimate. However, this presupposes that the information about affecting labels in Constraint 1 is indeed automatically captured by saturated post-hoc optimization as claimed above. We now show that this is the case if scf_h is the unique minimum saturated cost function for h and *cost*.

Theorem 2. *With unique minimum saturated cost functions, h^{SPhO} dominates h^{PhO}, i.e., $h^{\text{SPhO}}(cost, s) \geq h^{\text{PhO}}(cost, s)$ for all cost functions cost and states s.*

Proof. Comparing Definitions 4 and 5, the linear programs that define the two heuristics are identical except for the difference in coefficients in Constraint 1 and Constraint 5. For heuristic h and label ℓ, the coefficient in Constraint 5 ("SPhO coefficient") is $scf_h(\ell)$, while the corresponding coefficient in Constraint 1 ("PhO coefficient") is $cost(\ell)$ if $\ell \in \mathcal{A}(h)$ and 0 (the corresponding term does not appear) otherwise.

We show that the SPhO coefficient is always less than or equal to the corresponding PhO coefficient. From this it follows that Constraint 5 is at least as tight as Constraint 1, and consequently the set of feasible solutions for the SPhO-LP is a subset of the set of feasible solutions for the PhO-LP. For a minimization LP, this proves that the objective value for the SPhO-LP is greater or equal to the objective value of the PhO-LP, proving the theorem.

It remains to show the relationship between the coefficients for every heuristic h and label ℓ. If $\ell \in \mathcal{A}(h)$, we need to show $scf_h(\ell) \leq cost(\ell)$, which holds because it is one of the defining properties of saturated cost functions.

If $\ell \notin \mathcal{A}(h)$, we must show $scf_h(\ell) \leq 0$. We know that ℓ does not affect h because $\mathcal{A}(h)$ is a superset of the set of labels that affect h. Let $cost'$ be the (non-negative) cost function which is like $cost$ except that $cost'(\ell) = 0$. From the definition of "affects" and ℓ not affecting h, we obtain $h(cost', s) = h(cost, s)$ for all states s. Moreover, we have $cost'(\ell') \leq cost(\ell')$ for all labels ℓ' from the definition of $cost'$. These two properties show that $cost'$ is a saturated cost function for h and $cost$. Because scf_h is the *unique minimum* saturated cost function for h and $cost$, it never exceeds $cost'$. We obtain $scf_h(\ell) \leq cost'(\ell) = 0$, which concludes the proof. □

We note that the example in Figure 1 shows that there are cases where the dominance is strict: we have $h^{\text{SPhO}}(cost, s_1) = 8 > 5 = h^{\text{PhO}}(cost, s_1)$.

From the proof, we see that we do not necessarily need a unique minimum saturated cost function to establish dominance. It is sufficient to consider any saturated cost function that assigns a cost of at most 0 to all labels outside of $\mathcal{A}(h)$, and such a saturated cost function always exists by iterating the construction of $cost'$ in the proof over all labels $\ell \notin \mathcal{A}(h)$.

Other Cost Partitioning Algorithms

Saturated post-hoc optimization is the latest addition to an extensive collection of cost partitioning algorithms. Seipp, Keller, and Helmert (2017a) compare cost partitioning algorithms both theoretically and experimentally. They show for cost-monotonic heuristics that saturated cost partitioning dominates greedy zero-one cost partitioning (Haslum, Bonet, and Geffner 2005; Edelkamp 2006) and that by saturating costs in a uniform cost partitioning (Katz and Domshlak 2008) we obtain an *opportunistic* version of uniform cost partitioning that dominates the original.

Up until now, post-hoc optimization was the last cost partitioning algorithm without a direct connection to cost saturation. As we show above, the two ideas can indeed be combined into saturated post-hoc optimization, which dominates post-hoc optimization. Finally, since post-hoc optimization dominates the canonical heuristic (Haslum et al. 2007; Pommerening, Röger, and Helmert 2013), we can infer that saturated post-hoc optimization dominates the canonical heuristic as well.

This leaves the question of how h^{SPhO} compares to the other cost partitioning algorithms that are not already known to be dominated, i.e., saturated cost partitioning (h^{SCP}) and opportunistic uniform cost partitioning (h^{OUCP}). We already know that h^{SCP} and h^{OUCP} do not dominate each other (Seipp, Keller, and Helmert 2017a), and we show that there are no dominance relations between h^{SPhO} and h^{SCP} or h^{OUCP} either.

Theorem 3. *For each of the following pairs of cost partitioning algorithms there exists a weighted transition system $\langle \mathcal{T}, cost \rangle$ and a set of cost-monotonic heuristics $\mathcal{H}$ for $\mathcal{T}$*

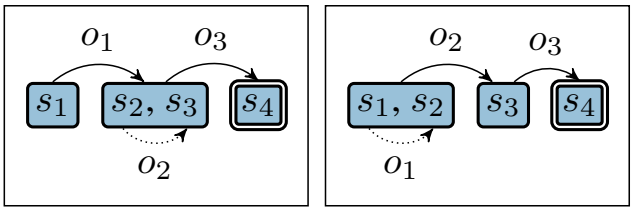

Figure 2: Example abstraction heuristics h_3 (left) and h_4 (right) used in the proof of Theorem 3. All operators cost 1. The s_i are concrete states, the rounded rectangles are abstract states, abstract goal states use double borders and dotted lines depict abstract self-loops.

such that

$$h^{\text{SCP}}_{\omega}(cost, s) > h^{\text{SPhO}}(cost, s) \tag{9}$$

$$h^{\text{SPhO}}(cost, s) > h^{\text{SCP}}_{\omega}(cost, s) \tag{10}$$

$$h^{\text{OUCP}}_{\omega}(cost, s) > h^{\text{SPhO}}(cost, s) \tag{11}$$

$$h^{\text{SPhO}}(cost, s) > h^{\text{OUCP}}_{\omega}(cost, s) \tag{12}$$

for a state $s \in S(\mathcal{T})$ and an order ω of $\mathcal{H}$.

Proof. Consider the abstraction heuristics h_1 and h_2 in Figure 1 for a weighted transition system $\langle \mathcal{T}, cost \rangle$. For state s_2 we have $h^{\text{SCP}}_{\langle h_1, h_2 \rangle}(cost, s_2) = 8 > h^{\text{SPhO}}(cost, s_2) = 7.2 > h^{\text{SCP}}_{\langle h_2, h_1 \rangle}(cost, s_2) = 7 > h^{\text{OUCP}}_{\langle h_1, h_2 \rangle}(cost, s_2) = 6$, showing (9), (10) and (12). As an example for (11), consider the abstraction heuristics h_3 and h_4 in Figure 2 for a weighted transition system $\langle \mathcal{T}', cost' \rangle$. For both heuristic orders ω, $h^{\text{OUCP}}_{\omega}(cost', s_1) = 3 > h^{\text{SPhO}}(cost', s_1) = 2$. □

Experiments

We implemented saturated post-hoc optimization in the Fast Downward planning system (Helmert 2006) and used the Downward Lab toolkit (Seipp et al. 2017) for running experiments on Intel Xeon Silver 4114 processors. Our benchmark set consists of all 1827 tasks without conditional effects from the optimal sequential tracks of the International Planning Competitions 1998–2018. We limit time by 30 minutes and memory by 3.5 GiB. All benchmarks, code and experiment data are published online (Seipp, Keller, and Helmert 2021).

In our experiments we compute cost partitionings over four different sets of abstraction heuristics: pattern databases found by hill climbing (HC, Haslum et al. 2007), systematic pattern databases of sizes 1 and 2 (SYS, Pommerening, Röger, and Helmert 2013), Cartesian abstractions of *landmark* and *goal* task decompositions (CART, Seipp and Helmert 2018) and the combination of these three heuristic sets (COMB=HC+SYS+CART).

Table 1 compares the number of solved tasks by h^{PhO} and h^{SPhO} for the four types of heuristic sets. We see that saturating the costs is beneficial for all considered abstraction heuristics: the overall coverage increases by 9, 51, 176 and 171 tasks, respectively. A per-domain analysis reveals that h^{SPhO} solves more tasks than h^{PhO} in 6, 17, 20 and 20 domains across the four settings, while the opposite is true in

	HC		SYS		CART		COMB	
agricola (20)	0		0		0		0	
airport (50)	23		28		25	**+2**	31	
barman (34)	4		4		4		4	
blocks (35)	28		26	**+1**	18		26	**+1**
childsnack (20)	0		0		0		0	
data-network (20)	10		9		10		12	
depot (22)	7		7		4	**+1**	7	
driverlog (20)	12		12		7		13	
elevators (50)	40		33		33	**+4**	42	
floortile (40)	2		2		2		2	
freecell (80)	17	**+1**	14	**+1**	14	**+39**	15	**+38**
ged (20)	19		15		15		15	
grid (5)	3		2		2		2	**+1**
gripper (20)	7		7		7		7	
hiking (20)	10		10	**+1**	10	**+1**	9	**+2**
logistics (63)	27		25		16	**+8**	28	
miconic (150)	63		50	**+2**	52	**+81**	60	**+82**
movie (30)	30		30		30		30	
mprime (35)	22		21		25		24	
mystery (30)	15		15		17		17	
nomystery (20)	19		16		10	**+5**	19	
openstacks (100)	40		38	**+2**	37	**+1**	37	**+2**
organic (20)	7		7		7		7	
organic-split (20)	9		7		8		7	
parcprinter (50)	28		30	**+2**	24		28	**+3**
parking (40)	**10**	-1	3	**+4**	0		2	**+4**
pathways (30)	4		4		4		4	
pegsol (50)	44		44	**+4**	44	**+2**	44	**+4**
petri-net (20)	0		2	**+2**	1	**+1**	0	
pipes-nt (50)	15	**+1**	15	**+1**	15	**+1**	16	**+1**
pipes-t (50)	10	**+2**	8	**+3**	10	**+1**	9	**+1**
psr-small (50)	49	**+1**	49	**+1**	48		49	**+1**
rovers (40)	7		6		5	**+1**	7	
satellite (36)	6		6		5		6	
scanalyzer (50)	23		11	**+16**	9	**+4**	15	**+12**
snake (20)	8	**+1**	6	**+1**	7		6	**+1**
sokoban (50)	44		48	**+1**	35	**+6**	48	**+1**
spider (20)	12		12	**+1**	7	**+4**	12	**+1**
storage (30)	15		15		14		15	
termes (20)	9		**6**	-1	5		8	
tetris (17)	5	**+4**	3		5		3	
tidybot (40)	21		18		7	**+8**	18	**+3**
tpp (30)	6		6		6		6	
transport (70)	31		21		22		27	
trucks (30)	7		7		6	**+4**	7	**+3**
visitall (40)	28		27		12		27	
woodwork (50)	15		25	**+9**	9	**+2**	25	**+9**
zenotravel (20)	13		11		8		12	**+1**
Sum	824	**+9**	761	**+51**	661	**+176**	808	**+171**

Table 1: Absolute number of solved tasks by traditional post-hoc optimization and the difference in coverage when using saturated post-hoc optimization instead for the four different types of abstraction heuristics.

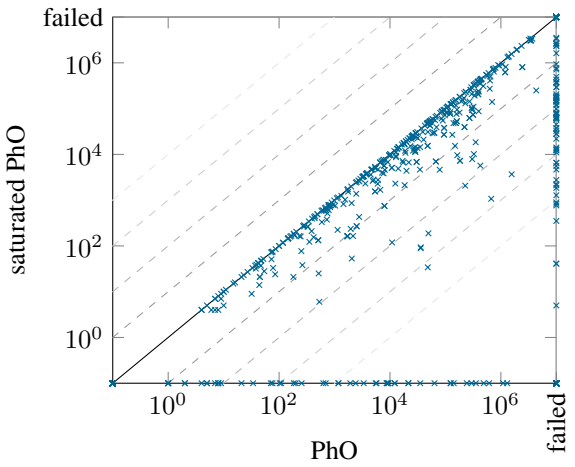

Figure 3: Expansions excluding the last f layer for traditional and saturated post-hoc optimization when using the combination of abstraction heuristics (COMB).

only a single domain for HC and SYS and never for CART nor COMB.

Figure 3 explains why h^{SPhO} solves so many more tasks than h^{PhO} for the combination of abstraction heuristics (COMB) by comparing the number of expansions before the last f layer: saturating the costs often makes the resulting heuristic much more accurate. There are 327 commonly solved tasks for which cost saturation decreases the number of expansions. Furthermore, for 52 of these 327 tasks h^{PhO} expands states before the last f layer whereas h^{SPhO} is perfect.

Conclusion

We showed that we can transport the idea of cost saturation to post-hoc optimization by considering for each heuristic only the fraction of costs that actually contributes to its estimates. The result is a cost partitioning algorithm that dominates the original both in theory and on the IPC benchmarks.

Acknowledgments

We have received funding for this work from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 817639). Moreover, this research was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215.

References

Edelkamp, S. 2001. Planning with Pattern Databases. In Cesta, A.; and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 84–90. AAAI Press.

Edelkamp, S. 2006. Automated Creation of Pattern Database Search Heuristics. In Edelkamp, S.; and Lomuscio, A., eds.,

Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006), 35–50.

Felner, A.; Korf, R.; and Hanan, S. 2004. Additive Pattern Database Heuristics. *Journal of Artificial Intelligence Research* 22: 279–318.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New Admissible Heuristics for Domain-Independent Planning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, 1163–1168. AAAI Press.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26: 191–246.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In Boddy, M.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 176–183. AAAI Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *Journal of the ACM* 61(3): 16:1–63.

Karpas, E.; and Domshlak, C. 2009. Cost-Optimal Planning with Landmarks. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1728–1733. AAAI Press.

Katz, M.; and Domshlak, C. 2008. Optimal Additive Composition of Abstraction-based Admissible Heuristics. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 174–181. AAAI Press.

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

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

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From Non-Negative to General Operator Cost Partitioning. In Bonet, B.; and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3335–3341. AAAI Press.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the Most Out of Pattern Databases for Classical Planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2357–2364. AAAI Press.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based Heuristics for Cost-optimal Planning. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 226–234. AAAI Press.

Schrijver, A. 1998. *Theory of linear and integer programming*. John Wiley & Sons.

Seipp, J.; and Helmert, M. 2013. Counterexample-guided Cartesian Abstraction Refinement. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 347–351. AAAI Press.

Seipp, J.; and Helmert, M. 2014. Diverse and Additive Cartesian Abstraction Heuristics. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 289–297. AAAI Press.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *Journal of Artificial Intelligence Research* 62: 535–577.

Seipp, J.; Keller, T.; and Helmert, M. 2017a. A Comparison of Cost Partitioning Algorithms for Optimal Classical Planning. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 259–268. AAAI Press.

Seipp, J.; Keller, T.; and Helmert, M. 2017b. Narrowing the Gap Between Saturated and Optimal Cost Partitioning for Classical Planning. In Singh, S.; and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3651–3657. AAAI Press.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated Cost Partitioning for Optimal Classical Planning. *Journal of Artificial Intelligence Research* 67: 129–167.

Seipp, J.; Keller, T.; and Helmert, M. 2021. Code, benchmarks and experiment data for the AAAI 2021 paper "Saturated Post-hoc Optimization for Classical Planning". https://doi.org/10.5281/zenodo.4302051.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.