Subset-Saturated Cost Partitioning for Optimal Classical Planning

Jendrik Seipp and **Malte Helmert**
University of Basel
Basel, Switzerland
{jendrik.seipp,malte.helmert}@unibas.ch

Abstract

Cost partitioning is a method for admissibly adding multiple heuristics for state-space search. Saturated cost partitioning considers the given heuristics in sequence, assigning to each heuristic the minimum fraction of remaining costs that it needs to preserve its estimates for all states. We generalize saturated cost partitioning by allowing to preserve the heuristic values of only a subset of states and show that this often leads to stronger heuristics.

Introduction

Solving challenging search problems optimally often requires using multiple heuristics (e.g., Holte et al. 2006). One way to combine heuristics is to use the maximum over their estimates. However, this only selects the single most accurate heuristic for each state. Cost partitioning is a general way to make *adding* multiple heuristics admissible by distributing the cost of each action among the heuristics.

The literature contains many cost partitioning algorithms (e.g., Haslum, Bonet, and Geffner 2005; Haslum et al. 2007; Katz and Domshlak 2008; 2010; Pommerening, Röger, and Helmert 2013). Recent experiments (Seipp, Keller, and Helmert 2017a) suggest that saturated cost partitioning (Seipp and Helmert 2014; 2018) is the cost partitioning method of choice for the benchmark set of the International Planning Competitions (IPC). Saturated cost partitioning operates on a cost function c and a sequence of heuristics $\mathcal{H}$. Beginning with the first heuristic h in the sequence, it computes the minimum cost function c' that h needs to preserve its estimates under c for all states. It then allocates this cost function c' to h and iterates by partitioning the remaining cost function $c - c'$ among the remaining heuristics in the sequence in the same way.

Choosing a cost function that preserves the heuristic value of all states is a very conservative choice that strongly favors heuristics that occur early in $\mathcal{H}$. The second and later heuristics in $\mathcal{H}$ only get to use costs that are of no possible utility for the earlier heuristics in the sequence. Consequently, the order of $\mathcal{H}$ strongly affects heuristic quality. Previous work acknowledges this by maximizing over multiple saturated cost partitionings derived from different orders (e.g., Seipp, Keller, and Helmert 2017b).

Preserving the heuristic values of all states is often wasteful. For example, if we can afford to compute a dedicated cost partitioning for each state s encountered during search, there is no need to preserve any heuristic values besides $h(s)$. In this paper, we generalize saturated cost partitioning to preserve the heuristic values of a subset of states, which can be the set of all states (as in previous work), a single state, or anything in between. We then consider the theoretical and computational properties of this generalization for the case of abstraction heuristics.

It turns out that such "subset-saturated" cost partitionings have a more complex structure than in the case where the heuristic values of all states are preserved. In particular, previous results on the existence of unique minimal saturated cost partitionings do not hold in the generalized setting. Instead, minimal saturated cost partitionings for subsets of states form a Pareto frontier with no obvious "best" cost function, leaving a large design space for algorithms that compute suitable cost functions.

We therefore conduct a more general study of cost partitioning methods involving trade-offs between computation time and the sets of states whose heuristic values are preserved. Our experiments show that a planning system using the new cost partitioning algorithms compares favorably with earlier cost partitioning algorithms and other state-of-the-art planning systems on the IPC benchmark suite.

Transition Systems

Cost partitioning can be applied to any collection of heuristics for state-space search, and therefore our definitions are not specific to classical planning. We first define transition systems, which are also known as state spaces.

Definition 1. ***Transition Systems.***
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 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* $S_\star(\mathcal{T})$ *of* goal states.

The objective in state-space search is to find paths from the initial state to a goal state.

Definition 2. ***Paths and Goal Paths.***
Let $\mathcal{T}$ be a transition system. A path *from* $s \in S(\mathcal{T})$ *to* $s' \in$

$S(\mathcal{T})$ is a sequence of transitions from $T(\mathcal{T})$ of the form $\pi = \langle s^0 \xrightarrow{\ell_1} s^1, \ldots, s^{n-1} \xrightarrow{\ell_n} s^n \rangle$, where $s^0 = s$ and $s^n = s'$. The length *of π, denoted by $|\pi|$, is n. The empty path (of length 0) is permitted if $s = s'$. A* goal path *from $s \in S(\mathcal{T})$ is a path from s to any goal state $s' \in S_\star(\mathcal{T})$.*

So far, we have not introduced a notion of (label or path) *cost*. This does not mean that we consider the unit-cost setting but rather that cost functions must be provided *in addition to* the transition system. This separation makes it easier to consider the same transition system with varying cost functions, which is a key concept for cost partitioning.

Definition 3. ***Cost Functions.***
A cost function *for transition system $\mathcal{T}$ is a function cost : $L(\mathcal{T}) \to \mathbb{R} \cup \{-\infty, \infty\}$. A cost function cost is* finite *if $-\infty < cost(\ell) < \infty$ for all labels ℓ. It 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 $\mathcal{T}$ and $\mathcal{C}_{\geq 0}(\mathcal{T})$ for the set of non-negative cost functions for $\mathcal{T}$.

We speak of *general* cost functions when we want to emphasize that a cost function is not required to be non-negative or finite. Allowing infinite costs in cost functions goes beyond previous work and is necessary to cleanly state some of our formal results. Negative costs were already considered in previous work (Pommerening et al. 2015).

Allowing infinities means that we must take care in arithmetic expressions that involve both $+\infty$ and $-\infty$. We will consider two different kinds of addition. *Left addition* is denoted by the regular summation operators $+$ (infix) and $\sum$ (prefix) and handles infinities as $\infty + x = \infty$ and $-\infty + x = -\infty$ for all x, including $x \in \{\infty, -\infty\}$. In particular, sums involving both kinds of infinities evaluate to the leftmost infinite value in the sum. This operation is associative, but not commutative. We will use left addition to combine multiple heuristic estimates within cost partitioning.

Path addition is denoted by the operators $\oplus$ (infix) and $\bigoplus$ (prefix) and handles infinities as $x \oplus y = \infty$ iff $x = \infty$ or $y = \infty$, and $x \oplus (-\infty) = -\infty \oplus x = -\infty$ for all $x \neq \infty$. In other words, sums involving mixed infinities evaluate to $+\infty$. This operation is associative and commutative. Path addition is used to combine the costs of multiple transitions along a path. We will interpret transitions of cost $-\infty$ as "infinitely cheap" and transitions of cost ∞ as "non-existent", which explains why $-\infty \oplus \infty = \infty$: a path that uses a non-existent transition cannot really be used and therefore has infinite cost even if it uses an infinitely cheap transition. Of course, this is just an intuitive interpretation. The formal reason for these definitions is that they allow the usual theorems on properties of heuristics and cost partitioning to generalize to cases involving infinite costs.

With finite values, both kinds of summation follow the usual rules of addition. Note that both operations behave identically when at least one of the two operands is finite. In fact, in sums involving two operands, the only difference is that $-\infty + \infty = -\infty$, while $-\infty \oplus \infty = \infty$.

Definition 4. ***Weighted Transition Systems.***
A weighted transition system *is a pair $\langle \mathcal{T}, cost \rangle$ where $\mathcal{T}$ is a transition system and cost is a cost function for $\mathcal{T}$.*

As usual, we extend cost functions from labels to paths.

Definition 5. ***Cost of a Path.***
The cost *of a path $\pi = \langle s^0 \xrightarrow{\ell_1} s^1, \ldots, s^{n-1} \xrightarrow{\ell_n} s^n \rangle$ in a weighted transition system $\langle \mathcal{T}, cost \rangle$ is defined as $cost(\pi) = \bigoplus_{i=1}^{n} cost(\ell_i)$.*

Note that by our definition of path addition ($\bigoplus$), the cost of any path including a label of cost ∞ is ∞, even if the path also includes labels of cost $-\infty$.

We can now define the notion of optimal paths.

Definition 6. ***Goal Distances and Optimal Paths.***
The goal distance *of a state $s \in S(\mathcal{T})$ in a weighted transition system $\langle \mathcal{T}, cost \rangle$ is defined as $\inf_{\pi \in \Pi_\star(\mathcal{T}, s)} cost(\pi)$, where $\Pi_\star(\mathcal{T}, s)$ is the set of goal paths from s in $\mathcal{T}$. (The infimum of the empty set is ∞.)*

We write $h^*_{\mathcal{T}}(cost, s)$ for the goal distance of s in $\langle \mathcal{T}, cost \rangle$ and omit $\mathcal{T}$ from the notation where the transition system does not matter or is clear from context.

A goal path π from s is optimal *under the given cost function if $cost(\pi) = h^*_{\mathcal{T}}(cost, s)$.*

We define the goal distances as an infimum rather than a minimum because it is possible that no minimum exists if there are negative-cost cycles in the transition system.

In general, we have $h^*(cost, s) \in \mathbb{R} \cup \{-\infty, \infty\}$, so goal distances can be negative or infinite. If *cost* is non-negative, then so is h^*. However, finite *cost* does not imply finite h^*: $h^*(cost, s) = -\infty$ can follow from cycles with negative finite cost and $h^*(cost, s) = \infty$ from lack of goal paths.

In *optimal classical planning*, we are given a compact description of a transition system and a finite non-negative cost function, and the objective is to find an optimal goal path for the initial state or show that no such goal path exists.

Heuristics

Heuristics are functions that estimate goal distance (Pearl 1984). The literature usually defines heuristics as functions of states, i.e., for a fixed cost function. We define them as functions of cost functions and states so that we can introduce the notion of cost partitioning cleanly.

Definition 7. ***Heuristics, Admissibility and Consistency.***
A heuristic *for a transition system $\mathcal{T}$ is a function $h : \mathcal{C}(\mathcal{T}) \times S(\mathcal{T}) \to \mathbb{R} \cup \{-\infty, \infty\}$.*

Heuristic h is admissible *if $h(cost, s) \leq h^*_{\mathcal{T}}(cost, s)$ for all $cost \in \mathcal{C}(\mathcal{T})$ and all $s \in S(\mathcal{T})$.*

Heuristic h is consistent *if $h(cost, s) \leq cost(\ell) \oplus h(cost, s')$ for all $cost \in \mathcal{C}(\mathcal{T})$ and all $s \xrightarrow{\ell} s' \in T(\mathcal{T})$.*

Heuristics are used in *heuristic search algorithms* like A* (Hart, Nilsson, and Raphael 1968) to find optimal goal paths. These algorithms generally require admissible heuristics to guarantee optimality of solutions, and consistency is usually desirable to avoid extra work due to *reopening* of states.

Heuristics in the literature are usually defined for finite non-negative cost functions. It is not necessarily obvious how their definition and properties like admissibility generalize to arbitrary cost functions. In all cases where we consider specific heuristics in this paper, these are *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 mapping 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: $h(\mathit{cost}, s) = h^*_{\mathcal{T}'}(\mathit{cost}, \alpha(s))$. We introduced h^* for general cost functions in Definition 6, so abstraction heuristics for general cost functions are well-defined. Abstraction heuristics for finite non-negative cost functions are admissible and consistent (e.g., Helmert, Haslum, and Hoffmann 2007). It is easy to verify that admissibility and consistency generalize to arbitrary cost functions (Seipp and Helmert 2019).

Note, however, that generalizing *implementations* of abstraction heuristics is more challenging than generalizing their definition. For non-negative cost functions, abstract goal distances can be computed with Dijkstra's (1959) algorithm, which can be implemented with worst-case runtime $O(N \log N + M)$, where $N = |S(\mathcal{T}')|$ and $M = |T(\mathcal{T}')|$ (Cormen, Leiserson, and Rivest 1990). For possibly negative cost functions, no algorithms are known that substantially outperform the Bellman-Ford algorithm (Bellman 1958) in the worst case, whose worst-case runtime is $O(NM)$.

If we consider a (not particularly large) abstract transition system with $N = 10000$ states and at least $N \log N$ transitions, this means that the Bellman-Ford algorithm is 10000 times slower than Dijkstra's algorithm (ignoring the constant factors hidden in the big-O notation). We will come back to the question what this means for the practical use of abstraction heuristics with general cost functions later.

Cost Partitioning

Challenging state-space search problems often require using multiple heuristics that capture different parts of the problem (Holte et al. 2006). *Cost partitioning* (Katz and Domshlak 2008; 2010) is a general approach for combining multiple admissible heuristics into a single admissible heuristic.

Definition 8. ***Cost Partitioning.***
Let $\mathcal{T}$ be a transition system. A cost partitioning *for a cost function cost $\in \mathcal{C}(\mathcal{T})$ is a tuple $\langle \mathit{cost}_1, \dots, \mathit{cost}_n \rangle \in \mathcal{C}(\mathcal{T})^n$ whose sum is bounded by cost: $\sum_{i=1}^{n} \mathit{cost}_i(\ell) \leq \mathit{cost}(\ell)$ for all $\ell \in L(\mathcal{T})$.*

A cost partitioning algorithm *for a transition system $\mathcal{T}$ and a tuple of heuristics $\mathcal{H} = \langle h_1, \dots, h_n \rangle$ takes a cost function cost as its input and produces a cost partitioning $\langle \mathit{cost}_1, \dots, \mathit{cost}_n \rangle$ for cost as its output. It induces the* cost-partitioned heuristic $h(\mathit{cost}, s) = \sum_{i=1}^{n} h_i(\mathit{cost}_i, s)$.

Note that according to our definition of left addition ($\sum$), the sum of multiple (positive or negative) infinite heuristic estimates is the first infinite term.

Cost-partitioned heuristics derived from admissible (consistent) component heuristics are admissible (consistent). Cost partitioning forms the basis of most state-of-the-art heuristics in optimal classical planning (e.g., Karpas and Domshlak 2009; Helmert and Domshlak 2009; Pommerening, Röger, and Helmert 2013; Seipp and Helmert 2014; Pommerening et al. 2015; Seipp, Keller, and Helmert 2017a; Seipp 2017).

Katz and Domshlak (2008) studied cost partitioning in the case where the overall cost function *cost* and component cost functions cost_i are finite and non-negative. Pommerening et al. (2015) generalized this by allowing negative costs in cost_i (but not *cost*). It is easy to verify that the admissibility and consistency results also apply to our more general definition (Seipp and Helmert 2019).

It is possible to compute an optimal cost partitioning for a given state (i.e., a cost partitioning resulting in the largest possible heuristic value for the given state) in polynomial time for abstraction (Katz and Domshlak 2008; 2010) and landmark (Karpas and Domshlak 2009) heuristics. These results were initially proved for non-negative cost functions, later generalized to possibly negative cost functions (Pommerening et al. 2015), and it is not difficult to further generalize them to handle infinite costs. However, the computation usually takes too much time and/or memory to be feasible for challenging planning tasks (e.g., Pommerening, Röger, and Helmert 2013; Seipp, Keller, and Helmert 2017b; Seipp 2018).

Subset-Saturated Cost Partitioning

Saturated cost partitioning (Seipp and Helmert 2014; 2018) is a greedy algorithm that quickly computes suboptimal cost partitionings. It has been shown to perform significantly better experimentally than other techniques such as optimal cost partitioning, uniform cost partitioning, greedy zero-one cost partitioning, and the canonical heuristic for pattern databases (Seipp, Keller, and Helmert 2017a). Since we already described the algorithm in the introduction, we only briefly recapitulate it here. We also provide a pseudo-code definition of a generalized algorithm below (Definition 12).

Saturated cost partitioning works on a set of heuristics processed sequentially. The cost function allocated to the first heuristic is chosen in such a way that all heuristic values under this new cost function are identical to the heuristic values under the original cost function. Among all cost functions that satisfy this property, we select one where all label costs are as low as possible. The leftover costs are then used to recursively cost-partition the rest of the sequence.

Preserving the heuristic values of all states can be quite wasteful. For example, some states might not be reachable from the initial state, so that an algorithm like A* would never consider their heuristic values. Instead of wasting costs on uninteresting states, we may want to only preserve the heuristic values of a subset of states in order to retain a larger portion of the label costs for other heuristics. The following two definitions formalize this idea.

Definition 9. ***Dominating Cost Functions.***
Consider two cost functions cost and cost$'$ defined on the same set of labels. We say that cost dominates *cost$'$, in symbols cost $\leq$ cost$'$, if cost$(\ell) \leq$ cost$'(\ell)$ for all labels ℓ.*

In words, dominance is the usual elementwise partial order on functions. This is a weak notion of dominance: $\mathit{cost} \leq \mathit{cost}'$ does not imply that the two functions are different.

Definition 10. ***Saturated Cost Functions.***
Consider a transition system $\mathcal{T}$, a set of states $S' \subseteq S(\mathcal{T})$, 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 S', h and cost if*

1. $\mathit{scf} \leq \mathit{cost}$ and

2. $h(\mathit{scf}, s) = h(\mathit{cost}, s)$ for all states $s \in S'$.

A saturated cost function scf is minimal *if no other saturated cost function for S', h and cost dominates it.*

If scf is non-negative, it is minimal among non-negative cost functions *if no other non-negative saturated cost function for S', h and cost dominates it.*

The definition generalizes our previous definition (Seipp and Helmert 2014), which considered the special case $S' = S(\mathcal{T})$. A saturated cost function may not exceed the given cost function *cost* and must preserve the heuristic values of all states in the given subset. Such a function always exists: *cost* itself satisfies both properties.

It is less obvious whether *minimal* saturated cost functions always exist and whether such minima are unique when they exist. Cost functions can be incomparable, so that in general the minimal saturated cost functions form a Pareto set.

The key step in saturated cost partitioning is to compute a saturated cost function for a given cost function. We call a function that performs this computation a *saturator*.

Definition 11. ***Saturators.***
Consider a transition system $\mathcal{T}$, a set of states $S' \subseteq S(\mathcal{T})$ and a heuristic h for $\mathcal{T}$.

A saturator *for S' and h is a partial function $\mathit{saturate} : \mathcal{C}(\mathcal{T}) \to \mathcal{C}(\mathcal{T})$ such that whenever $\mathit{saturate}(\mathit{cost})$ is defined, it is a saturated cost function for S', h and cost.*

A saturator is general *if its domain of definition is $\mathcal{C}(\mathcal{T})$. It is* non-negative to general *(*NNG*) if its domain of definition is $\mathcal{C}_{\geq 0}(\mathcal{T})$. It is* non-negative *if its domain of definition is $\mathcal{C}_{\geq 0}(\mathcal{T})$ and it only produces cost functions in $\mathcal{C}_{\geq 0}(\mathcal{T})$.*

Saturators have not previously been introduced in the literature: earlier work exploited results on the uniqueness of minimal saturated cost functions to simply speak of "the" minimal saturated cost function. These uniqueness results do not hold in the more general setting we consider.

However, saturators implicitly exist in earlier work. The paper that introduced saturated cost partitioning (Seipp and Helmert 2014) only considered non-negative saturators. This was later generalized to NNG saturators (Keller et al. 2016). General saturators have not been previously considered in the literature. In the following we assume that saturators are general unless stated otherwise.

We can now formally define our generalization of saturated cost partitioning. The main differences to earlier definitions (e.g., Seipp and Helmert 2018) are that we consider general cost functions, parameterize the definition by a saturator and allow saturators for subsets of states.

Definition 12. ***Subset-Saturated Cost Partitioning.***
Consider a transition system $\mathcal{T}$, a set of states $S' \subseteq S(\mathcal{T})$, a non-empty sequence of heuristics $\mathcal{H} = \langle h_1, \dots, h_n \rangle$ for $\mathcal{T}$ and a sequence $\mathit{Saturate} = \langle \mathit{saturate}_1, \dots, \mathit{saturate}_n \rangle$ such that $\mathit{saturate}_i$ is a saturator for S' and h_i for all $1 \leq i \leq n$.

The saturated cost partitioning *$\langle \mathit{cost}_1, \dots, \mathit{cost}_n \rangle$ of the cost function cost induced by Saturate is defined as:*

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

where the auxiliary cost functions remain_i represent the remaining costs after processing the first i heuristics in $\mathcal{H}$.

The subtraction in the definition of remain_i follows the rules of left addition and the definition $a - b := a + (-b)$. Hence, if $\mathit{remain}_{i-1}(\ell)$ is (positively or negatively) infinite, then we always obtain $\mathit{remain}_i(\ell) = \mathit{remain}_{i-1}(\ell)$. In particular, when the leftover costs for a label are ∞, we may allocate cost ∞ to all further cost functions because $\infty - \infty = \infty$ under left addition.

It is easy to see that the saturated cost partitioning is indeed a cost partitioning (Definition 8), i.e., $\sum_{i=1}^{n} \mathit{cost}_i(\ell) \leq \mathit{cost}(\ell)$ for all labels ℓ. For labels ℓ with $\mathit{cost}(\ell) = \infty$ this holds trivially. For labels ℓ with $\mathit{cost}(\ell) = -\infty$, we must have $\mathit{remain}_i(\ell) = -\infty$ for all $0 \leq i \leq n$ because $-\infty - x = -\infty$ for all x. Hence we get $\mathit{cost}_i(\ell) = -\infty$ for all $1 \leq i \leq n$ because cost_i is bounded by remain_{i-1} by the definition of saturated cost functions, which shows $\sum_{i=1}^{n} \mathit{cost}_i(\ell) = -\infty = \mathit{cost}(\ell)$. (This uses $n \geq 1$.)

It remains to consider the case where $\mathit{cost}(\ell)$ is finite. If all $\mathit{cost}_i(\ell)$ are finite or $-\infty$, the cost partitioning property is easy to show, so consider the case where $\mathit{cost}_i(\ell) = \infty$ for some $1 \leq i \leq n$. Let i_0 be the smallest index with this property. Then we must have $\mathit{remain}_{i-1}(\ell) = \infty$. With finite $\mathit{cost}(\ell)$, this is only possible if $\mathit{cost}_j(\ell) = -\infty$ for some $j < i_0$, which implies $\sum_{i=1}^{n} \mathit{cost}_i(\ell) = -\infty < \mathit{cost}(\ell)$.

In addition to the set of heuristics, there are two major choice points in using saturated cost partitioning: firstly, due to its greedy nature, the cost partitioning highly depends on the order in which the heuristics are considered (Seipp, Keller, and Helmert 2017b; Seipp 2017). Secondly, the choice of saturators is clearly important. In the next section, we introduce four saturators, discuss some of their theoretical properties and evaluate them experimentally.

When comparing saturators, we are particularly interested in dominance results between saturated cost functions because dominating cost functions are "more economical" than the cost functions they dominate: they achieve the same objective of preserving heuristic values while leaving a larger portion of the available costs to the later heuristics in the sequence. Due to the greediness of saturated cost partitioning, this does not necessarily translate into better overall heuristics, but our experiments show that it usually does. The following general result is useful to establish dominance:

Theorem 1. ***Domination for $S'' \subseteq S'$.***
For a given transition system $\mathcal{T}$, heuristic h for $\mathcal{T}$ and cost function $\mathit{cost} \in \mathcal{C}(\mathcal{T})$, let $\mathit{SCF}(X)$ be the set of saturated cost functions for the set of states $X \subseteq S(\mathcal{T})$, h and cost.

We say that a cost function cost is the unique minimum *of a set of cost functions Cost if it dominates all cost functions*

in Cost. (Not all sets Cost have a unique minimum.)
Let $S'' \subseteq S' \subseteq S(\mathcal{T})$. Then:

1. *For all cost functions $cost' \in SCF(S')$, there exists a cost function $cost'' \in SCF(S'')$ that dominates $cost'$.*
2. *If $cost''$ is the unique minimum of $SCF(S'')$, then $cost''$ dominates all cost functions in $SCF(S')$.*

A corresponding result also holds for saturated cost functions that are minimal among non-negative cost functions rather than minimal in general.

To prove either result, it is sufficient to observe that whenever requirement 2 for saturated cost functions (Definition 10) holds for a given state set S', it also holds for all subsets $S'' \subseteq S'$, and therefore $SCF(S'') \supseteq SCF(S')$.

The practical significance of the theorem is that it shows that saturators for $S'' \subseteq S'$ are more economical than saturators for S'. However, we must be careful to note that this is a dominance results for *sets*: even minimal saturated cost functions for $SCF(S'')$ are not guaranteed to dominate *all* saturated cost functions for $SCF(S')$. This stronger notion of dominance is only guaranteed if $SCF(S'')$ has a unique minimum (part 2 of the theorem).

In summary, one way to obtain a dominating (more economical) saturated cost function is to consider a subset of states. A second way is to *compose* saturators, i.e., apply one saturator to the output of another. We formalize this result in the following theorem.

Theorem 2. ***Domination by Composing Saturators.*** *Let $saturate_1$ and $saturate_2$ be general saturators for the same transition system $\mathcal{T}$, state set S' and heuristic h. Let $saturate_{12} : \mathcal{C}(\mathcal{T}) \to \mathcal{C}(\mathcal{T})$ be the composition of these saturators, i.e., $saturate_{12}(cost) = saturate_2(saturate_1(cost))$ for all cost functions $cost \in \mathcal{C}(\mathcal{T})$.*

Then $saturate_{12}$ is a general saturator for $\mathcal{T}$, S' and h, and for all cost functions $cost \in \mathcal{C}(\mathcal{T})$, $saturate_{12}(cost)$ dominates $saturate_1(cost)$.

In other words, we can "improve" any saturator (make it more economical) by applying another saturator to its result.

The theorem follows directly from requirement 1 for saturated cost functions: the composed saturator consists of the outer saturator applied to the inner saturator, and the result of every saturator must dominate its input by requirement 1.

The same result holds for non-negative saturators and non-negative cost functions, but not for NNG saturators because in this case the inner saturator $saturate_1$ can produce a possibly negative cost function which $saturate_2$ cannot process. The ability to compose saturators in this way is the main reason why we introduced general saturators in Definition 11 rather than limiting the definition to the previously considered NNG saturators.

For NNG saturators, we can use a modified form of composition: $saturate_{12} = saturate_2(\max(saturate_1(cost), 0))$, where 0 is the constant-zero function and $\max$ is element-wise maximum. In words, negative costs produced by the inner saturator are replaced by 0. The resulting function is a saturator because all original costs for an NNG saturator are at least 0, so raising the output of $saturate_1$ to 0 does not violate requirement 1 of Definition 10. However, the dominance result of Theorem 2 does not hold in this case.[1]

Saturators for Abstraction Heuristics

In the following, we introduce and analyze several saturators for (explicitly represented) abstraction heuristics. The general theory in the previous sections is not limited to abstractions, but the practical computation of suitable saturators for other classes of heuristics (other than landmark heuristics, which can easily be compiled into abstraction heuristics) is an open research question not addressed in this paper.

Saturate for All States (*all*)

We start with the saturator that preserves the heuristic estimates for all states, as in previous work. We will call this saturator *all* in the following. In previous work (Seipp and Helmert 2018), we show how to compute a cost function $mscf \leq cost$ (both in the non-negative and NNG case) that preserves all heuristic estimates and that is a unique minimum among all such cost functions. The key idea is to make sure that for each label ℓ, the consistency constraint $h(mscf, s) \leq mscf(\ell) + h(mscf, s')$ is tight for at least one state transition $s \xrightarrow{\ell} s'$. In the NNG case, this can be enforced by setting

$$mscf(\ell) = \sup_{\substack{a \xrightarrow{\ell} b \in T(\mathcal{T}') \\ \text{s.t. } h^*_{\mathcal{T}'}(cost, a) < \infty}} (h^*_{\mathcal{T}'}(cost, a) - h^*_{\mathcal{T}'}(cost, b)),$$

where $\mathcal{T}'$ is the abstract transition system underlying h. (The supremum of the empty set is $-\infty$.) For the non-negative cost setting, replace all negative costs in the result by 0.

To turn this idea into a general saturator, we need to handle negative and infinite label costs in the input. This can be achieved by applying three preprocessing steps to $\mathcal{T}'$ before computing *mscf*. Firstly, for all labels ℓ with $cost(\ell) = \infty$, set $mscf(\ell) = \infty$, remove all transitions labeled with ℓ from the transition system, and continue the computation of *mscf* as if ℓ did not exist.[2] Secondly, remove all abstract states with $h^*_{\mathcal{T}'}(cost, a) = -\infty$ and their incident transitions. (Note that such states can also arise in the finite-cost case due to negative cost cycles.) These do not need to be considered because $h^*_{\mathcal{T}'}(cost, a) = -\infty$ implies $h^*_{\mathcal{T}'}(cost', a) = -\infty$ for all cost functions $cost' \leq cost$ because goal distances monotonically depend on the cost function. Hence, their heuristic value will be preserved by *every* cost function bounded from above by *cost*. Finally, remove all abstract states and incident transitions from which there is no path to the goal. For these the heuristic value is ∞ independently of the cost function, so again they do not need

[1] In the next section we consider the *perim* saturator, which cannot handle finite negative costs in its input, but *can* handle $-\infty$. In this case, we replace finite negative costs by 0 but preserve $-\infty$.

[2] Strictly speaking, this may assign a higher cost to ℓ than necessary, and thus the resulting cost function might not be minimal. But this is never an issue because there is no need to save costs when $cost(\ell) = \infty$, as the remaining costs will be ∞ no matter how we set $mscf(\ell)$, as $\infty - \infty = \infty$ under left addition. Thus, fully utilizing infinite costs is an optimal choice.

to be considered when computing *mscf*. After these transformations, all remaining goal distances in $\mathcal{T}'$ are finite, and we can apply the original algorithm.

Saturate for Reachable States (*reach*)

In a forward search, there is no point in preserving heuristic values of states that cannot be reached from the initial state, as these values are never used during search. In an abstraction heuristic, we can overapproximate the set of reachable states by the preimage S' of all abstract states that are reachable in the abstract transition system.

Our second saturator, denoted by *reach*, is a saturator for this set S'. It can be computed by pruning all unreachable states from the abstract transition system $\mathcal{T}'$ (since we do not want to preserve their values) and then applying the *all* saturator to the result. For the same reason as for *all*, this results in the unique minimum saturated cost function for S'.

We will consider two different variants of *reach* in our experiments. In the *online* setting, where we compute a different saturated cost partitioning for each state s encountered during search, we evaluate reachability with respect to s. In the *offline* setting, where we precompute saturated cost partitionings prior to search that are then used for all states encountered during search, we evaluate reachability with respect to the initial state. In this way, in either setting *reach* only ever disregards states that do not matter.

We remark that the same general idea of pruning the "uninteresting" states from $\mathcal{T}'$ does *not* work for arbitrary sets S'. For example, we can clearly not compute a saturated cost function that preserves just the initial state by pruning all abstract states other than the abstract initial state. Unless the abstract initial state is an abstract goal state, this would result in an empty transition system and hence in the constant saturated cost function $-\infty$. The critical property that makes this idea work for S' is that goal paths from reachable states can only pass through reachable states. A similar *goal path closure* property will also be important for the next saturator.

Saturate for a Perimeter (*perim*)

So far, we have considered saturators that preserve all potentially useful heuristic values. The remaining two saturators are less conservative. The perimeter saturator (*perim*) preserves all heuristic values within a given perimeter of the goal. It is based on the idea, well-established in heuristic search (e.g., Holte et al. 2004; 2006; Torralba, Linares López, and Borrajo 2018), that it is more important for heuristic estimates to be accurate close to the goal than far away from the goal.

Given an abstraction heuristic h based on abstract transition system $\mathcal{T}'$ and a numerical parameter $k \geq 0$, consider the set of states within a perimeter of k of the goal, i.e., with heuristic values of at most k: $S'_k = \{s \in S \mid h(\mathit{cost}, s) \leq k\}$. In the abstract transition system, this corresponds to all abstract states a with $h^*_{\mathcal{T}'}(\mathit{cost}, a) \leq k$.[3] If *cost* is a non-negative function, this set satisfies a somewhat weaker variant of the goal path closure property: goal paths starting in states in S'_k *may* pass through states outside the perimeter, but they must eventually return to a state on the boundary of S'_k and then stay inside S'_k until they reach a goal state.

In the setting of non-negative cost functions, this closure property is sufficient for the same idea that we used for *reach* (restrict $\mathcal{T}'$ to S'_k, then apply the *all* saturator to the result) to result in a unique minimum saturated cost function for S'_k in the case where all label costs are 0 or 1. However, this uniqueness property does not hold in general.

For general non-negative cost functions, things are slightly more complicated for the same reason that algorithms like perimeter search (Dillenburg and Nelson 1994) are more complicated for non-unit-cost problems. In this case, a saturated cost function for S'_k can be computed by applying the *reach* saturator to a modified transition system, where in addition to the states from S'_k we include all abstract states a with $h(\mathit{cost}, a) > k$ for which there exist abstract state transitions $a \xrightarrow{\ell} b$ with $h(\mathit{cost}, b) < k < h(\mathit{cost}, a)$ that "pierce" the perimeter, along with all such transitions (but not other transitions incident to such states a). We then treat these added states as if they had a heuristic value of k. This modification ensures that the states on the boundary of S'_k cannot be bypassed.

For negative costs, the situation is yet more complicated, and we conjecture that computing a perimeter with an effective closure property requires at least as much work as finding shortest paths in directed graphs with possibly negative edge costs. This can become prohibitive even for moderate-size abstractions, and hence we only consider non-negative cost functions as inputs for *perim*. (As an exception, we do support label cost $-\infty$ because it can be removed in preprocessing as discussed for *all*.)

In our experiments, we set the parameter k to $h(\mathit{cost}, s)$ when using the *perim* saturator for heuristic h. Here, *cost* is the cost function to which *perim* is applied, and s is the currently evaluated state (in the online setting) or the sample state for which the cost partitioning is optimized (in the offline setting; see the section describing the experiments for more details).

Saturate for a Single State (*lp*)

We conclude our discussion of saturators by considering the least conservative case of only preserving the heuristic value for a singleton state set $\{s\}$. In an online setting with a dedicated cost partitioning for each state, such a saturator can potentially result in the best heuristic values, as it does not spend costs on anything other than the evaluated state.

Single-state saturation is more complex than the earlier saturators because there is no unique minimum, not even in the non-negative finite case. Consider a saturator for $\{s\}$ in a

[3]In general, this set of abstract states might be larger than necessary to cover S'_k because there may exist abstract states with no concrete preimage, and such abstract states would not need to be considered. All our experiments are based on *induced abstractions*, in which the abstraction mapping α is surjective and therefore this case cannot arise. The perimeter saturator is still applicable without this restriction, but considering more abstract states than necessary means potentially missing an opportunity to preserve more costs.

transition system with two goal paths for s, one using the labels ℓ_1 and ℓ_2 and another using the label ℓ_3. All labels have unit cost. Then all cost functions *cost*$'$ with *cost*$'(\ell_3) = 1$ and *cost*$'(\ell_1) +$ *cost*$'(\ell_2) = 1$ are minimum saturated cost functions for $\{s\}$. In general, the minimum saturated cost functions form a large Pareto frontier with no obvious mechanism to select from the frontier. Many of these minima do not even dominate the much more conservative saturator *all*, as we will see in the experiments in the following section. One remedy for this is to compose saturators (Theorem 2). By first applying *all* and then the single-state saturator, we can guarantee dominance over *all*.

As a consequence of the complex solution structure, there is no obvious greedy way of computing a minimum saturated cost function for a single state. We conjecture that the problem is at least as hard as finding shortest paths in digraphs with possibly negative arc weights.

We solve the problem by linear programming and call the resulting saturator *lp*. We first restrict the transition systems to all states that lie on paths between s and a goal state and remove all infinities, as in *all*. It is then easy to verify that the set of saturated cost functions for *cost* and $\{s\}$ for the abstract transition system $\mathcal{T}'$ and abstraction function α can be characterized by the following linear constraints:

$$\mathsf{H}_a \leq 0 \qquad \text{for all } a \in S_\star(\mathcal{T}') \tag{1}$$

$$\mathsf{H}_a \leq \mathsf{C}_\ell + \mathsf{H}_b \qquad \text{for all } a \xrightarrow{\ell} b \in T(\mathcal{T}') \tag{2}$$

$$\mathsf{C}_\ell \leq \mathit{cost}(\ell) \qquad \text{for all } \ell \in L(\mathcal{T}') \tag{3}$$

$$\mathsf{H}_{\alpha(s)} = h(s) \tag{4}$$

The variables C_ℓ encode the saturated cost function, and the variables H_a represent h^* values in $\mathcal{T}'$ under this cost function. (Using a standard LP trick, H_a may be an underestimation of the true h^*.) A Pareto-optimal solution can then be extracted by solving the LP with a suitable objective function. We choose to minimize $\sum_{\ell \in L} \mathsf{C}_\ell$ where L is the set of labels occurring in any constraint. Any other linear combination of these variables with positive coefficients would also result in a Pareto-optimal solution. The costs of all labels not mentioned in any constraints are set to $-\infty$.

We remark that the same approach can be used to saturate for arbitrary subsets of states S' by restricting the transition system to all states between S' and the goal and using the same LP, but with one constraint of type (4) for each $s \in S'$.

Negative Costs and the Offline Setting

We conclude this section by discussing a computational concern. In general, subset-saturated cost partitioning requires the ability to evaluate a given heuristic under an arbitrary cost function. Before processing heuristic h_i, we must evaluate it under the *remain*$_i$ cost function, and after computing *cost*$_i$, we may have to reevaluate h_i under the *cost*$_i$ function. The first reevaluation is generally not too costly because *remain*$_i$ is always a non-negative cost function (for non-negative original cost functions, as in classical planning). However, *cost*$_i$ can in general include negative costs.

This has not been an issue in earlier work on saturated cost partitioning, which used the *all* saturator that guarantees that $h_i(\mathit{remain}_i, s) = h_i(\mathit{cost}_i, s)$ for all states s, so no recomputation is required. It is also not an issue in the online setting where we only need the heuristic value of one state s, which is preserved by the cost partitioning. However, it is a concern when we saturate a heuristic for a state subset S' and later evaluate it on states $s \notin S'$. When negative costs are permitted, this requires algorithms like Bellman-Ford to reevaluate the heuristic. We found this to be prohibitively expensive in experiments. Therefore, we do not actually compute $h_i(\mathit{cost}_i, s)$ but rather use a lower bound that can be directly extracted from the computations of the saturators, trading off heuristic quality for computation time.

For all states preserved by the saturators, the exact heuristic values are known. For states where the preprocessing of infinities detects that the heuristic value must be ∞ or $-\infty$, we use these values. For *reach*, we use $-\infty$ for all unreachable states, since these heuristic values will never be evaluated anyway. For *perim* with a perimeter radius of k, we use a heuristic value of k for all states outside the perimeter. Finally, for *lp* we use the values of the LP variables H_a for all abstract states present in the LP and $-\infty$ for all others. All these values can be extracted with very little overhead while computing the saturated cost functions.

Experiments

We implemented all saturators in the Fast Downward planning system (Helmert 2006) and conducted experiments with the Downward Lab toolkit (Seipp et al. 2017) on Intel Xeon Silver 4114 processors. Our benchmark set consists of all 1827 tasks without conditional effects from the optimization tracks of the 1998–2018 IPCs. We use a time limit of 30 minutes and a memory limit of 3.5 GiB.

We compute saturated cost partitionings over the same set of abstraction heuristics as in earlier work (Seipp 2018): pattern databases found by hill climbing (Haslum et al. 2007), systematic pattern databases (Pommerening, Röger, and Helmert 2013) and Cartesian abstractions of landmark and goal task decompositions (Seipp and Helmert 2018). We order the heuristics according to the greedy ordering method with the $\frac{h}{\mathit{stolen}}$ scoring function, the best ordering in previous work on saturated cost partitioning (Seipp 2018). This ordering method takes a state as an input and attempts to order the heuristics in a way that is good for this state. The ordering does not depend on which saturator is used. All benchmarks[4], code[5] and experimental data[6] have been published online.

Online Subset-Saturated Cost Partitioning

We first consider the online setting where we compute a saturated cost partitioning for each state evaluated during an A* search. (This also includes computing a potentially different heuristic order for each state.) In addition to the four basic saturators *all*, *reach*, *perim* and *lp*, we consider compositions, which we write in forward composition notation. For example, "*reach*, *perim*, *lp*" means to first apply the *reach* saturator, then *perim* on its result, then *lp* on that result.

[4]Benchmarks: https://doi.org/10.5281/zenodo.2616479

[5]Code: https://doi.org/10.5281/zenodo.2616502

[6]Experimental data: https://doi.org/10.5281/zenodo.2616510

	all	reach	perim	reach$^{(+)}$, perim	lp	all, lp	reach, lp	perim, lp	reach$^{(+)}$, perim, lp
all	–	2	2	2	88	2	2	2	2
reach	**15**	–	5	2	88	2	2	2	2
perim	**433**	**430**	–	0	142	15	15	6	6
reach, perim	**436**	**434**	**6**	–	142	15	15	6	6
lp	**493**	**490**	**229**	**226**	–	79	79	84	84
all, lp	**507**	**505**	**219**	**216**	**179**	–	**0**	12	12
reach, lp	**507**	**505**	**219**	**216**	**179**	**0**	–	12	12
perim, lp	**512**	**510**	**217**	**214**	**185**	**16**	**16**	–	**0**
reach, perim, lp	**512**	**510**	**217**	**214**	**185**	**16**	**16**	**0**	–
all	–	2	65	65	**407**	5	5	53	53
reach	**48**	–	75	69	**412**	11	4	58	57
perim	**325**	**314**	–	0	**505**	109	92	3	3
reach$^+$, perim	**332**	**319**	**21**	–	**507**	113	95	4	3
lp	282	277	169	168	–	107	100	110	110
all, lp	**367**	**359**	**236**	**234**	**505**	–	6	117	117
reach, lp	**378**	**370**	**241**	**238**	**521**	**28**	–	**119**	**118**
perim, lp	**407**	**397**	**199**	**188**	**542**	**133**	117	–	0
reach$^+$, perim, lp	**408**	**397**	**200**	**188**	**542**	**134**	117	**1**	–

Table 1: Per-task comparison of heuristic estimates for the initial state. Top: non-negative saturators. Bottom: general saturators. The entry in row r and column c shows the numbers of tasks where saturator r returns a higher initial state estimate than saturator c. We only consider the 1506 (non-negative) and 1390 (general) tasks for which all nine configurations compute the initial state heuristic value within the time and memory limits. We highlight the maximum of the entries (r, c) and (c, r) in bold.

Using Theorems 1 and 2 we can restrict the set of useful combinations. For example, *all* only makes sense at the beginning of a composition and *lp* only at the end, and many combinations are redundant. For example, "*all*, *reach*" would be equivalent to *reach*. This still leaves a large set of possibilities, of which we consider a reasonable subset.

The upper part of Table 1 compares the quality of the resulting heuristics by showing in how many benchmarks a given non-negative saturator results in a higher initial heuristic value than another non-negative saturator. We see that ignoring unreachable states (*reach*) has a mild benefit over considering all states (*all*). Saturating for the perimeter (*perim*) has a very significant advantage over *all* and *reach*. Comparisons with the *lp* saturator have the largest variance: it is very often better than the other saturators, but also often worse, even when compared to *all*. This can be explained by the lack of unique minimum when saturating for a single state, as different cost functions from the Pareto frontier can behave wildly differently.

	Coverage		Evals/sec		$h(s_0)$ higher	
	nn	gen	nn	gen	nn	gen
all	680	**703**	940.9	**946.2**	7	**195**
reach	657	**679**	501.4	**514.1**	5	**222**
perim	722	**726**	897.7	**906.6**	4	**99**
reach$^{(+)}$, perim	689	**695**	415.1	**423.1**	4	**111**
lp	**360**	320	**8.7**	6.9	**493**	144
all, lp	**385**	354	**10.0**	8.1	138	**180**
reach, lp	**384**	355	**9.9**	8.1	120	**181**
perim, lp	**392**	367	**11.0**	8.8	31	**134**
reach$^{(+)}$, perim, lp	**395**	366	**10.8**	8.7	30	**134**

Table 2: Comparison of non-negative (nn) and general (gen) saturators in the online setting. Coverage: Number of solved tasks. Evals/sec: geometric mean of evaluations per second for the 131 commonly solved tasks that need at least 100 evaluations. $h(s_0)$ higher: number of tasks where non-negative/general saturators yield a higher $h(\mathit{cost}, s_0)$ than their counterparts among the 1389 tasks for which all heuristics report $h(\mathit{cost}, s_0)$ in both settings.

Composing multiple saturators clearly pays off, underlining the dominance result of Theorem 2. In particular, applying *lp* after other saturators stabilizes its behavior. The best configuration *reach*, *perim*, *lp* produces a better heuristic estimate than the previous state of the art *all* in 512 cases, while being worse in only 2.

The corresponding results for general saturators are shown in the lower part of Table 1.[7] Most trends are similar to the non-negative scenario, but there is much more variance. Most strikingly, *lp* is worse than the other basic saturators more often than not, showing that selecting appropriately from the Pareto frontier is even more of an issue for general cost functions. Combining *reach* and/or *perim* with *lp* results in the strongest heuristics.

Table 2 shows coverage results and state evaluation rates for planning algorithms using these saturators. We see that an increase in heuristic accuracy does not always lead to solving more tasks. As a reminder, these algorithms compute a new cost partitioning for each evaluated state. Saturator *reach* has a noticeable runtime overhead, and the overhead of *lp* is so large that this saturator is clearly not worth using in this context. In contrast, since *perim* is almost as fast to evaluate as *all* (Table 2) and yields better estimates (Table 1), it solves more tasks than *all* in both the non-negative and the general setting.

Table 2 also shows that all non-LP-based saturators perform significantly better in terms of heuristic quality, runtime and coverage when producing general cost functions

[7]The *reach*$^+$ saturator is a version of *reach* that replaces all finite negative costs with 0 but preserves $-\infty$. This modification is necessary in compositions with *perim* because, as discussed in the previous section, *perim* cannot handle finite negative costs.

	all	reach	perim	lp	perim*	Coverage
all	–	**1**	**9**	**36**	1	1136
reach	0	–	**8**	**36**	1	1134
perim	1	2	–	**36**	0	1117
lp	0	0	0	–	0	694
perim*	**4**	**4**	**11**	**36**	–	**1144**

Table 3: Per-domain coverage comparison of saturated cost partitioning heuristics computed offline using different saturators. The entry in row r and column c shows the number of domains in which saturator r solves more tasks than saturator c. For each saturator pair we highlight the maximum of the entries (r, c) and (c, r) in bold. Right: Total number of solved tasks.

than when producing non-negative ones. Therefore, we only consider general saturators in the remaining experiments.

Offline Subset-Saturated Cost Partitioning

As shown by the low state evaluation rates in the online experiment, saturated cost partitioning appears to be more useful as an *offline* technique, where one or more cost partitionings are precomputed and then used for all states encountered during search. This is the setting in which saturated cost partitioning has previously been considered (e.g., Seipp, Keller, and Helmert 2017b). Therefore, we now report results for precomputed cost partitionings.

We follow our previous approach (Seipp, Keller, and Helmert 2017b), changing only the saturators used. We start with an empty family $\mathcal{F}$ of cost-partitioned heuristics and sample 1000 *evaluation states* $\hat{S}$ with random walks (Haslum et al. 2007). We then iteratively sample a new state s called an *optimization target* in the same way, compute a heuristic order and saturated cost partitioning optimized for s and add the resulting cost-partitioned heuristic h to $\mathcal{F}$ if there exists an evaluation state $\hat{s} \in \hat{S}$ for which $h(\hat{s})$ is larger than for all heuristics already in $\mathcal{F}$. For both the evaluation states and the optimization targets, the first generated state is the initial state rather than a state sampled by random walk. We stop this *diversification* procedure after 200 seconds and then perform an A* search using the maximum over the heuristics in $\mathcal{F}$.

Unlike the online setting, for saturators other than *all* and *reach*, the precomputed heuristics can be evaluated on states whose heuristic values the saturators in question made no attempt to preserve. Saturated cost partitioning using the *perim* saturator will spend no label costs on states outside the perimeter, even if this means that label costs remain completely unused. This is clearly undesirable and easy to address. If any costs are left over after a full cost partitioning run using *perim*, we can start another cost partitioning run with the remaining costs using the *all* saturator, which always exploits all exploitable costs. This results in two cost functions for each component heuristic, which can simply be added. We call the resulting approach *perim** = *perim* + *all*.

	Comp1	Comp2	PPDBs	perim$^\star_{\mathrm{M}}$
Coverage	1028	1100	1085	**1195**
#Domains perim$^\star_{\mathrm{M}}$ better	28	26	27	–
#Domains perim$^\star_{\mathrm{M}}$ worse	12	15	13	–

Table 4: Total and per-domain coverage comparison of state-of-the-art planners.

Table 3 shows the results of this experiment. We present overall coverage and the number of domains in which algorithm X solves more tasks than algorithm Y for all pairs (X, Y). We see that the *lp* saturator is clearly too slow even when used in this offline setting and the runtime penalty for *reach* also makes it perform slightly worse than *all*. The *perim* saturator by itself leaves too much cost unused for good overall performance, but in the *perim** variant, it leads to an improvement over the previous state of the art *all*.

The strong performance of *perim** raises the question how this algorithm fares against the state of the art in optimal classical planning. Table 4 compares *perim** to the top three non-portfolio planners from IPC 2018, Complementary1 (Franco et al. 2018), Complementary2 (Franco et al. 2017) and Planning-PDBs (Moraru et al. 2018). Since all three IPC planners prune irrelevant operators in a preprocessing step (Alcázar and Torralba 2015), Table 4 evaluates a version of *perim** that uses the same technique, denoted by *perim*$^\star_{\mathrm{M}}$. The results show that *perim*$^\star_{\mathrm{M}}$ has an edge over the three IPC planners in a per-domain comparison and solves the highest number of tasks in total.

Conclusion

We generalized the saturated cost partitioning algorithm by preserving the heuristic estimates only for a subset of states. To evaluate our generalization we introduced several saturators for computing saturated cost functions. Both in the online setting, where we compute a cost partitioning for each evaluated state, and the offline setting, where we precompute a set of cost-partitioned heuristics before search, the subset saturators yield stronger heuristics than the earlier saturators that preserve all heuristic values.

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).

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In *Proc. ICAPS 2015*, 2–6.

Bellman, R. E. 1958. On a routing problem. *Quarterly of Applied Mathematics* 16:87–90.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms*. The MIT Press.

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

Dillenburg, J. F., and Nelson, P. C. 1994. Perimeter search. *AIJ* 65:165–178.

Edelkamp, S. 2001. Planning with pattern databases. In *Proc. ECP 2001*, 84–90.

Franco, S.; Torralba, Á.; Lelis, L. H. S.; and Barley, M. 2017. On creating complementary pattern databases. In *Proc. IJCAI 2017*, 4302–4309.

Franco, S.; Lelis, L. H. S.; Barley, M.; Edelkamp, S.; Martines, M.; and Moraru, I. 2018. The Complementary1 planner in the IPC 2018. In *IPC-9 planner abstracts*, 28–31.

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

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proc. AAAI 2007*, 1007–1012.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proc. AAAI 2005*, 1163–1168.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, 162–169.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proc. ICAPS 2007*, 176–183.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Holte, R.; Newton, J.; Felner, A.; Meshulam, R.; and Furcy, D. 2004. Multiple pattern databases. In *Proc. ICAPS 2004*, 122–131.

Holte, R. C.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *AIJ* 170(16–17):1123–1136.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *Proc. IJCAI 2009*, 1728–1733.

Katz, M., and Domshlak, C. 2008. Optimal additive composition of abstraction-based admissible heuristics. In *Proc. ICAPS 2008*, 174–181.

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

Keller, T.; Pommerening, F.; Seipp, J.; Geißer, F.; and Mattmüller, R. 2016. State-dependent cost partitionings for Cartesian abstractions in classical planning. In *Proc. IJCAI 2016*, 3161–3169.

Moraru, I.; Edelkamp, S.; Martinez, M.; and Franco, S. 2018. Planning-PDBs planner. In *IPC-9 planner abstracts*, 69–73.

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 *Proc. AAAI 2015*, 3335–3341.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In *Proc. IJCAI 2013*, 2357–2364.

Seipp, J., and Helmert, M. 2014. Diverse and additive Cartesian abstraction heuristics. In *Proc. ICAPS 2014*, 289–297.

Seipp, J., and Helmert, M. 2018. Counterexample-guided Cartesian abstraction refinement for classical planning. *JAIR* 62:535–577.

Seipp, J., and Helmert, M. 2019. Subset-saturated cost partitioning for optimal classical planning: Additional details. Technical Report CS-2019-001, University of Basel, Department of Mathematics and Computer Science.

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

Seipp, J.; Keller, T.; and Helmert, M. 2017a. A comparison of cost partitioning algorithms for optimal classical planning. In *Proc. ICAPS 2017*, 259–268.

Seipp, J.; Keller, T.; and Helmert, M. 2017b. Narrowing the gap between saturated and optimal cost partitioning for classical planning. In *Proc. AAAI 2017*, 3651–3657.

Seipp, J. 2017. Better orders for saturated cost partitioning in optimal classical planning. In *Proc. SoCS 2017*, 149–153.

Seipp, J. 2018. *Counterexample-guided Cartesian Abstraction Refinement and Saturated Cost Partitioning for Optimal Classical Planning*. Ph.D. Dissertation, University of Basel.

Torralba, Á.; Linares López, C.; and Borrajo, D. 2018. Symbolic perimeter abstraction heuristics for cost-optimal planning. *AIJ* 259:1–31.