# Subset-Saturated Transition Cost Partitioning for Optimal Classical Planning

**Anonymous**


### Abstract

Cost Partitionings allow additively combining the heuristic estimates of multiple abstractions. This often gives better heuristic estimates compared to a single abstraction heuristic. Subset-Saturated Cost Partitioning considers the abstraction heuristics in sequence and assigns the minimum fraction of remaining costs that it needs to preserve the heuristic estimates of a subset of states. We generalize Subset-Saturated Cost Partitioning by taking the context of applied actions into account. An empirical evaluation shows that the additional computational overhead caused by taking into account the application context of each action often does not pay off in terms of runtime, but leads to more informed heuristics.


## Introduction

Heuristic search with a lower bounding heuristic is one of the most promising techniques to solve challenging planning problems optimally. We consider additive cost partitioned abstraction heuristics that combine information from multiple sources (Katz and Domshlak 2008). A Cost Partitioning distributes the costs among multiple abstraction heuristics in a way that allows additively combining their heuristic estimates. Operator Cost Partitionings (Katz and Domshlak 2008) distribute costs over operators. Transition Cost Partitionings (Keller et al. 2016a; Pommerening et al. 2015) distribute costs over transitions allowing to take the context of applied actions into account.

Seipp and Helmert (2013a) showed that cost saturation is an effective approach to compute cost partitioned abstraction heuristics (Seipp, Keller, and Helmert 2017). It considers the abstraction heuristics in sequence and assigns the minimum fraction of the remaining costs that it needs to preserve the heuristic estimates. Recently, Seipp and Helmert (2019) proposed Subset-Saturated (Operator) Cost Partitioning that computes Operator Cost Partitionings and preserves the heuristic estimates of a subset of states that look more important during the search. Therefore, leaving more remaining costs for later abstractions.

In this work we define Subset-Saturated Transition Cost Partitioning, which additionally takes the context of applied actions into account. We use the Saturated Transition Cost Partitioning algorithm by Keller et al. (2016a) and preserve the heuristic estimates of the same subset of states given by Seipp and Helmert (2019). The benefit compared to Subset-Saturated Operator Cost Partitioning is that it never leaves any remaining costs unused. The drawback of this approach is that it is necessary to store costs of exponentially many transitions which adds decent computational overhead especially when there is no compact representation in the data structure based on decision diagrams (Keller et al. 2016a).

Our empirical evaluation shows that Subset-Saturated Transition Cost Partitioning computes better heuristic estimates in many tasks. But compared to Subset-Saturated Operator Cost Partitioning, the computation of heuristics with Subset-Saturated Transition Cost Partitioning often takes too much time, such that the improvements in heuristic quality do not pay off. This raises the question for future work whether and how it is possible to obtain more informative heuristics by considering the application context of actions in a way that the gain in heuristic quality pays off in practice.

## Background

In this chapter we define, describe and discuss the concepts and ideas that form the foundation for Subset-Saturated Transition Cost Partitioning.

### Transition Systems

A transition system describes the dynamics of a state-based system. Transition systems are also called state spaces.

**Definition 1** (Transition System (Seipp and Helmert 2019))**.** *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 finite set $T(\mathcal{T})$ of labeled transitions $s \xrightarrow{l} s$ with $s, s' \in S(\mathcal{T})$ and $l \in L(\mathcal{T})$, an initial state $s_0(\mathcal{T}) \in S(\mathcal{T})$ and a set $S_\star(\mathcal{T}) \subseteq S(\mathcal{T})$ 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 (Seipp and Helmert 2019))**.** *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 $\pi = \left\langle s^0 \xrightarrow{l_1} s^1, \ldots, s^{n-1} \xrightarrow{l_n} s^n \right\rangle$, where $s^0 = s$ and*

*$s^n = s'$. The length of $\pi$ denoted by $|\pi|$, is $n$. The empty path (of length 0) is permitted if $s = s'$. $\pi$ is called a goal path, if it is a path to any goal state $s' \in S_\star(\mathcal{T})$.*

In the context of Classical Planning and Cost Partitioning, it is convenient to allow assignment of costs to transitions that are not necessarily unit costs. Transition cost functions allow a different assignment for each transition.

**Definition 3** (Transition Cost Function)**.** *Let $\mathcal{T}$ be a transition system with transitions $T(\mathcal{T})$. A transition cost function for $\mathcal{T}$ is a function $tcf : T(\mathcal{T}) \to \mathbb{R} \cup \{-\infty, \infty\}$ that assigns costs to transitions. A transition cost function is finite if $-\infty < tcf(t) < \infty$ for all transitions $t \in T(\mathcal{T})$. It is nonnegative if $0 \leq tcf(t)$ for all transitions $t \in T(\mathcal{T})$. We write $\mathcal{C}(T(\mathcal{T}))$ for the set of all transition cost functions for $\mathcal{T}$ and $\mathcal{C}_{\geq 0}(T(\mathcal{T}))$ for the set of all nonnegative transition cost functions for $\mathcal{T}$.*

We speak of a general transition cost function if the transition cost function is not required to be nonnegative or finite. A special case of a transition cost function is an operator cost function (Seipp and Helmert 2019). It is a transition cost function where every transition with the same label is assigned the same cost value in the considered codomain. Hence, an operator cost function is a mapping from labels to the considered codomain. Note that an operator cost function could also be named a label cost function.

**Definition 4** (Operator Cost Function)**.** *Let $\mathcal{T}$ be a transition system with labels $L(\mathcal{T})$. An operator cost function for $\mathcal{T}$ is a function $ocf : L(\mathcal{T}) \to \mathbb{R} \cup \{-\infty, \infty\}$ that assigns costs to labels. We write $\mathcal{C}(L(\mathcal{T}))$ for the set of all operator cost functions for $\mathcal{T}$ and $\mathcal{C}_{\geq 0}(L(\mathcal{T}))$ for the set of all nonnegative operator cost functions for $\mathcal{T}$.*

Every operator cost function transforms into a transition cost function by setting the cost of every transition to the cost that is assigned to the label by the given operator cost function. The inverse transformation does not work because a transition cost function is more expressive than an operator cost function. Thus, we allow the implicit use of an operator cost function if we speak about a transition cost function.

In the context of Cost Partitioning, it is convenient to work with a collection of transition systems where each transition system is associated with a transition cost function. The following definition makes it more convenient to work with such a pairing (Seipp and Helmert 2019).

**Definition 5** (Weighted Transition System)**.** *A weighted transition system is a tuple $\langle \mathcal{T}, tcf \rangle$ where $\mathcal{T}$ is a transition system and $tcf$ is a transition cost function for $\mathcal{T}$.*

Allowing infinities to be assigned to transitions through either operator or transition cost functions means that we must take care of arithmetic expressions that involve $+\infty$ and $-\infty$ to make the theory of cost partitioning work. Seipp and Helmert (2019) defined two kinds of addition which makes it possible to handle mixed infinities in Cost Partitioning.

The symbols $+$ (infix) and $\sum$ (prefix) denote the left-addition operation. Left-addition over integers is the usual addition. Expressions that contain infinities are defined as $\infty + x = \infty$ and $-\infty + x = -\infty$ for all integers $x$, including $x$ being $\infty$ or $-\infty$. This operation is associative but not commutative. Intuitively, a left addition sum that contains mixed infinities evaluates to the leftmost infinity. In Cost Partitioning, we use left-addition for summing up multiple heuristic values and distributed costs (Seipp and Helmert 2019).

The symbols $\oplus$ (infix) and $\bigoplus$ (prefix) denote the path-addition operation. Path addition over integers is the usual addition. Expressions that contain infinites are defined as $x \oplus y = \infty$ iff $x = \infty$ or $y = \infty$ and $x \oplus (-\infty) = -\infty$ for all $x \neq \infty$. This operation is associative and commutative. Intuitively, a path addition sum evaluates to $+\infty$ if the path addition sum involves at least one $+\infty$. We use path-addition to define the cost of a path in a transition system (Seipp and Helmert 2019).

**Definition 6** (Cost of a path)**.** *Let $\langle \mathcal{T}, tcf \rangle$ be a weighted transition system. The cost of a path $\pi = \left\langle s^0 \xrightarrow{l_1} s^1, \ldots, s^{n-1} \xrightarrow{l_n} s^n \right\rangle$ with $t_i = s^{i-1} \xrightarrow{l_i} s^i$ is defined as $cost(tcf, \pi) = \bigoplus_{i=1}^{n} tcf(t_i)$.*

Intuitively, a path of cost $-\infty$ is infinitely cheap, and a path of cost $\infty$ is non-existent.

In Optimal Classical Planning, we are interested in finding paths with the cheapest cost to a goal. The following definition captures goal distances as functions that depend on a given state and a transition cost function. This notion follows the pairing of transition systems and transition cost functions in Cost Partitioning.

**Definition 7** (Goal Distances and Optimal Paths (Seipp and Helmert 2019))**.** *Let $\langle \mathcal{T}, tcf \rangle$ be a weighted transition system. The goal distance of a state $s \in S(\mathcal{T})$ in $\mathcal{T}$ under cost function $tcf$ is defined as $\inf_{\pi \in \Pi_\star(\mathcal{T}, s)} cost(tcf, \pi)$ where $\Pi_\star(\mathcal{T}, s)$ is the set of goal paths from $s$ in $\mathcal{T}$.*

*We write $h^*_{\mathcal{T}}(tcf, s)$ for the goal distance of $s$ in $\mathcal{T}$ under transition cost function $tcf$.*

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

The infimum ensures that repeatedly taking a negative cost cycle in the transition system will evaluate to $-\infty$.

## Planning Tasks

We consider SAS$^+$ like planning tasks (Bäckström and Nebel 1995) with state-independent action costs. A SAS$^+$ planning task with state independent costs is a 5-tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_\mathcal{I}, s_\star, c \rangle$ where $\mathcal{V}$ is a finite set of state variables and each variable $v$ has a finite domain $\mathcal{D}_v$. Every variable $v$ induces a set of facts of the form $(v, d)$ where $d \in \mathcal{D}_v$. A partial assignment $s$ is a set of facts of different variables. If $s$ contains a fact for each variable then $s$ is called a state. The set of all states over variables $\mathcal{V}$ is denoted by $S(\mathcal{V})$. For a state $s$ we write $s(v) = d$ if the state contains the fact $(v, d)$. A finite set of operators (or actions) $\mathcal{O}$ where each operator $o \in \mathcal{O}$ is a tuple $o = \langle \text{pre}(o), \text{eff}(o) \rangle$ and $\text{pre}(o), \text{eff}(o)$ are partial assigments. The set of operators induce a successor function $succ : \mathcal{O} \times S(\mathcal{V}) \to S(\mathcal{V})$ where $succ(o, s) = s'$ if $pre \subseteq s$ and $s'(v) = \mathit{eff}(v)$ if $\mathit{eff}(v)$ is defined and $s'(v) = s(v)$ otherwise. The initial state is $s_\mathcal{I}$ and the goal condition $s_\star$ is a partial assignment. The cost of

each operator is given by a state-independent cost function $c : \mathcal{O} \to \mathbb{R}_0^+$.

In state space search, we are traversing through the induced weighted transition system of a given planning task starting from the initial state. The components of the induced weighted transition system $\langle \mathcal{T}(\Pi), ocf \rangle$ for a SAS$^+$ planning task with state-independent costs $c : \mathcal{O} \to \mathbb{R}_0^+$ are defined as follows: The set of states equals the set of all states of $\Pi$ over $\mathcal{V}$, i.e. $S(\mathcal{T}) = S(\mathcal{V})$. The set of labels equals the set of operators, i.e. $L(\mathcal{T}) = \mathcal{O}$. There exists a transition with label $l \in L(\mathcal{T})$ from state $s \in S(\mathcal{T})$ to state $s' \in S(\mathcal{T})$ if the operator is applicable in $s$ and its effect transforms $s$ to $s'$, i.e. $T(\mathcal{T}) = \{s \xrightarrow{l} s' | s, s' \in S(\mathcal{T}) \wedge l \in L(\mathcal{T}) \wedge succ(l, s) = s'\}$. The initial state of the transition system equals the initial state of the planning task, i.e. $s_0(\mathcal{T}) = \mathcal{I}$. The set of goal states contains all states that satisfy the goal condition, i.e. $S_\star(\mathcal{T}) = \{s \in S(\mathcal{T}) | s \supseteq \gamma\}$. The cost of an operator with label $l \in L(\mathcal{T})$ equals the corresponding operator cost, i.e. $ocf(l) = cost(l)$ for all labels $l \in L(\mathcal{T})$.

## Abstractions

Abstractions are relaxations of the behavior of a state-based system where multiple states collapse into a single abstract state (Helmert, Haslum, and Hoffmann 2007).

**Definition 8** (Abstraction)**.** *Let $\mathcal{T}, \mathcal{T}'$ be two transition systems with the same label sets $L(\mathcal{T}) = L(\mathcal{T}')$ and let $\alpha : S(\mathcal{T}) \to S(\mathcal{T}')$, $\beta : T(\mathcal{T}) \to T(\mathcal{T}')$ be surjective functions. We say that $\mathcal{T}'$ is an abstraction of $\mathcal{T}$ with abstraction mappings $\alpha, \beta$ if (1) $\alpha(s_0(\mathcal{T})) = s_0(\mathcal{T}')$, (2) $\alpha(s_\star) \in S_\star(\mathcal{T}')$ for all $s_\star \in S_\star(\mathcal{T})$, and (3) $\alpha(s) \xrightarrow{l} \alpha(s') \in T(\mathcal{T}')$ and $\beta(s \xrightarrow{l} s') = \alpha(s) \xrightarrow{l} \alpha(s')$ for all $s \xrightarrow{l} s' \in T(\mathcal{T})$.*

We refer to $\mathcal{T}$ as the concrete transition system and $\mathcal{T}'$ as the abstract transition system. We consider a special type of abstraction where every abstract state is cartesian (Seipp and Helmert 2013b).

An abstraction heuristic is a function that maps each concrete state to the goal distance of its corresponding abstract state under a given transition cost function. Goal distances in the abstract transition system require an abstract transition cost function that maps every abstract transition to a value in the codomain. The abstract transition cost function maps an abstract transition $t'$ to the minimal cost of every concrete transition that maps to $t'$ (Keller et al. 2016a).

**Definition 9** (Abstraction heuristic)**.** *Let $\langle \mathcal{T}, tcf \rangle$ and $\langle \mathcal{T}', tcf' \rangle$ be two weighted transition systems. Let $\mathcal{T}'$ be an abstraction of $\mathcal{T}$ with abstraction mappings $\alpha, \beta$.*

*We say that $tcf'$ is the abstract transition cost function of $tcf$ that describes the cost of each abstract transition $t' \in T(\mathcal{T}')$ in the abstraction with*

$$tcf'(t') = \min\{tcf(t) \mid t \in T(\mathcal{T}) \wedge \beta(t) = t'\}$$

*The abstraction heuristic of a concrete state $s \in S(\mathcal{T})$ is the goal distance of the corresponding abstract state $\alpha(s)$ in the abstraction $\mathcal{T}'$ under the abstract transition cost function $tcf'$, i.e. $h(tcf, s) = h^*_{\mathcal{T}'}(tcf', \alpha(s))$.*

The definition of the abstract transition cost function reveals that the computation of an abstract transition cost involves a minimization over all (exponentially many) transitions of the concrete transition system that are responsible for this abstract transition (Geißer, Keller, and Mattmüller 2016). In the case of cartesian abstractions this minimization of exponentially many concrete transitions can be carried out in time that is often polynomial in the number of variables of the planning task, albeit worst-case exponential (Geißer, Keller, and Mattmüller 2016).

A heuristic is admissible if it never overestimates the goal distances in the weighted concrete transition system. Abstraction heuristics are admissible because every goal path in the abstract transition system corresponds to a goal path in the concrete transition system. The minimization in the abstract transition cost function ensures that the abstraction heuristic does not overestimate any goal distance (Keller et al. 2016a).

We use the $A^*$ algorithm with an admissible heuristic to find optimal goal paths (Hart, Nilsson, and Raphael 1968).

## Cost Partitionings

Cost Partitionings distribute costs among a set of abstractions in a way that allows admissibly adding their heuristic estimates.

**Transition Cost Partitioning** Transition Cost Partitioning partitions the available cost for each transition individually (Keller et al. 2016a; Pommerening 2017). This allows assigning a different degree of importance to different regions (or contexts) of the state space in different abstractions.

**Definition 10** (Transition Cost Partitioning)**.** *A transition cost partitioning for a weighted transition system $\langle \mathcal{T}, tcf \rangle$ with transition cost function $tcf \in \mathcal{C}(T(\mathcal{T}))$ is a tuple $\langle tcf_1, \ldots, tcf_n \rangle \in \mathcal{C}(T(\mathcal{T}))^n$ whose sum is bounded by $tcf$:*

$$\sum_{i=1}^{n} tcf_i(t) \leq tcf(t) \text{ for all } t \in T(\mathcal{T})$$

A Transition Cost Partitioning induces a heuristic by associating each transition cost function with an abstraction heuristic and summing up the heuristic estimates of individual states. The following theorem states that the resulting heuristic is admissible. Katz and Domshlak (2008) introduced Cost Partitioning that works on nonnegative operator cost functions. Pommerening et al. (2015) showed that general operator cost functions can be used for Cost Partitioning and Keller et al. (2016a) defined general transition cost function for Cost Partitioning. Finally, Seipp and Helmert (2019) introduced left addition rules to handle mixed infinities, which led to Theorem 1 and shows that cost partitioned abstraction heuristics are admissible.

**Theorem 1.** *Let $\mathcal{T}$ be a transition system with admissible heuristics $\langle h_1, \ldots, h_n \rangle$ and a transition cost partitioning $P(T(\mathcal{T})) = \langle tcf_1, \ldots, tcf_n \rangle \in \mathcal{C}(T(\mathcal{T}))^n$. Then $h_{P(T(\mathcal{T}))}(s) = \sum_{i=1}^{n} h_i(tcf_i, s)$ is an admissible heuristic.*

**Operator Cost Partitioning** Operator Cost Partitioning (Katz and Domshlak 2008) is a special case of Transition Cost Partitioning (Keller et al. 2016a) that partitions the available costs for each operator individually. Hence every transition cost function is restricted to be an operator cost function. Operator Cost Partitioning forms the basis of a state-of-the-art cost partitioned abstraction heuristic (Seipp and Helmert 2019).

**Definition 11** (Operator Cost Partitioning)**.** *An operator cost partitioning for a weighted transition system $\langle \mathcal{T}, ocf \rangle$ with operator cost function $ocf \in \mathcal{C}(L(\mathcal{T}))$ is a tuple $\langle ocf_1, \ldots, ocf_n \rangle \in \mathcal{C}(L(\mathcal{T}))^n$ whose sum is bounded by $ocf$:*

$$\sum_{i=1}^{n} ocf_i(l) \leq ocf(l) \text{ for all } l \in L(\mathcal{T})$$

There are different methods to compute Cost Partitionings. Algorithms based on cost saturation have shown to perform best on the IPC benchmark set (Seipp, Keller, and Helmert 2017).

## Subset-Saturated Cost Partitioning

Cost saturation is the key technique used in a state-of-the-art cost partitioned heuristic (Seipp and Helmert 2019). The idea of cost saturation is that a heuristic should be assigned just as much of the current remaining costs that it needs to preserve its estimates allowing to assign the remaining costs to other heuristics. The Saturated Cost Partitioning algorithm considers heuristics sequentially and applies cost saturation on every heuristic and the current remaining cost function. Finally, the cost functions assigned to each heuristic in the sequence are the saturated cost functions and form a Cost Partitioning. We speak of the Saturated Operator Cost Partitioning algorithm (Seipp, Keller, and Helmert 2020) if the procedure only considers operator cost functions and we speak of the Saturated Transition Cost Partitioning algorithm (Keller et al. 2016a) if it considers transition cost functions.

Saturated Cost Partitioning is sensitive to the heuristic order because an earlier heuristic might use costs that would be more useful in a later heuristic (Seipp 2017). Subset-Saturated (Operator) Cost Partitioning (Seipp and Helmert 2019) allows us to preserve the heuristic estimates of a subset of states, leaving more remaining costs for the following abstractions.

In this chapter, we generalize the concepts of Subset-Saturated Operator Cost Partitioning to Subset-Saturated Transition Cost Partitioning by dropping the constraint that cost functions must be operator cost functions and therefore allow them to be transition cost functions. Alternatively speaking, we allow preserving the heuristic estimates of a subset of states in the Saturated Transition Cost Partitioning algorithm. We begin with generalizing the notion of a dominating operator cost functions (Seipp and Helmert 2019) to transition cost functions. A transition cost function dominates another transition cost function if it never assigns a higher cost to a transition.

**Definition 12** (Dominating Transition Cost Function)**.** *Consider two transition cost functions $tcf$ and $tcf'$ defined on the same set of transitions. We say that $tcf$ dominates $tcf'$, in symbols $tcf \leq tcf'$, if $tcf(t) \leq tcf'(t)$ for all transitions $t$.*

*We say that $tcf$ is the unique minimum of a set of transition cost functions $Cost$ if it dominates all transition cost functions in $Cost$. (Not all sets $Cost$ have a unique minimum.)*

Dominating transition cost functions already occurred in the brief description of cost saturation, where the saturated cost function is a dominating cost function for the current remaining cost function. The generalization of a saturated operator cost function for a subset of states (Seipp and Helmert 2019) to transition cost functions is given in the following definition.

**Definition 13** (Saturated Transition Cost Function)**.** *Consider a weighted transition system $\langle \mathcal{T}, tcf \rangle$, a set of states $S' \subseteq S(\mathcal{T})$ and a heuristic $h$ for $\mathcal{T}$. A transition cost function $stcf \in \mathcal{C}(T(\mathcal{T}))$ is saturated for $S'$, $h$ and $tcf$ if*

1. *$stcf \leq tcf$ and*
2. *$h(stcf, s) = h(tcf, s)$, for all states $s \in S'$.*

We refer to a saturated transition cost function as a saturated operator cost function to emphasize that it is of its subtype. Observe that the transition cost function $tcf$ itself is a saturated transition cost function.

**Subset-Saturated Operator Cost Partitioning** Subset-Saturated Operator Cost Partitioning differs from Saturated Operator Cost Partitioning by an additional parameterization of an operator saturator that takes as an input an operator cost function, a heuristic and a subset of states of a transition system and outputs a saturated operator cost function for the subset of states (Seipp and Helmert 2019).

**Definition 14** (Operator Saturator)**.** *Consider a transition system $\mathcal{T}$, a set of states $S' \subseteq S(\mathcal{T})$ and a heuristic $h$ for $\mathcal{T}$.*

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

**Definition 15** (Subset-Saturated Operator Cost Partitioning)**.** *Consider a weighted transition system $\langle \mathcal{T}, ocf \rangle$, a set of states $S' \subseteq S(\mathcal{T})$, a nonempty sequence of heuristics $\mathcal{H} = \langle h_1, \ldots, h_n \rangle$ for $\mathcal{T}$ and a sequence $Saturate = \langle saturate_1, \ldots, saturate_n \rangle$ such that $saturate_i$ is an operator saturator for $S'$ and $h_i$ for all $1 \leq i \leq n$.*

*The operator cost partitioning $\langle ocf_1, \ldots, ocf_n \rangle$ of the operator cost function $ocf$ induced by $Saturate$ is defined as:*

$$remain_0 = ocf$$
$$ocf_i = saturate_{i-1}(remain_{i-1}) \text{ for all } 1 \leq i \leq n$$
$$remain_i = remain_{i-1} - ocf_i \text{ for all } 1 \leq i \leq n$$

For a detailed explanation of a set of operator saturators, we refer to the initial work on Subset-Saturated Operator Cost Partitioning by Seipp and Helmert (Seipp and Helmert 2019). We generalize these operator saturators in the following section and provide an additional saturator.

## Subset-Saturated Transition Cost Partitioning

We generalize Saturated Transition Cost Partitioning (Keller et al. 2016a) by allowing saturation for subsets similar to Subset-Saturated Operator Cost Partitioning (Seipp and Helmert 2019). The generalization can compute better heuristics, but due to the greedy nature of cost saturation, this is not always the case (Keller et al. 2016b).

We begin with generalizing operator saturators from Definition 14 to transition saturators that take as an input a transition cost function, a heuristic and a subset of states and outputs a saturated transition cost function for the subset of states.

**Definition 16** (Transition Saturator)**.** *Consider a transition system $\mathcal{T}$, a set of states $S' \subseteq S(\mathcal{T})$ and a heuristic $h$ for $\mathcal{T}$.*

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

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

The following definition generalizes Definition 15 by exchanging operator saturators with transition saturators.

**Definition 17** (Subset-Saturated Transition Cost Partitioning)**.** *Consider a weighted transition system $\langle \mathcal{T}, tcf \rangle$, a set of states $S' \subseteq S(\mathcal{T})$, a nonempty sequence of heuristics $\mathcal{H} = \langle h_1, \dots, h_n \rangle$ for $\mathcal{T}$ and a sequence $Saturate = \langle saturate_1, \dots, saturate_n \rangle$ such that $saturate_i$ is a saturator for $S' \subseteq S(\mathcal{T})$ and $h_i$ for all $1 \leq i \leq n$.*

*The transition cost partitioning $\langle tcf_1, \dots, tcf_n \rangle$ of the transition cost function $tcf$ induced by $Saturate$ is defined as:*

$$
\begin{aligned}
remain_0 &= tcf \\
tcf_i &= saturate_{i-1}(remain_{i-1}) \text{ for all } 1 \leq i \leq n \\
remain_i &= remain_{i-1} - tcf_i \text{ for all } 1 \leq i \leq n
\end{aligned}
$$

The following theorems generalize the domination results between operator saturators (Seipp and Helmert 2019) to transition saturators. The first theorem generalizes the domination by subsets to transition cost functions.

**Theorem 2.** *[Domination by Subsets] For a given transition system $\mathcal{T}$, heuristic $h$ for $\mathcal{T}$ and transition cost function $tcf \in \mathcal{C}(T(\mathcal{T}))$, let $STCF(X)$ be the set of saturated transition cost functions for the set of states $X \subseteq S(\mathcal{T})$, $h$ and $tcf$.*

*Let $S'' \subseteq S' \subseteq S(\mathcal{T})$. Then:*

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

**Proof:** Statement 1 follows from Definition 13 where we allow to assign lower saturated costs to transitions that start at or end at states outside the subset if it does not conflict with preserving the heuristic value of states within the subset. Such transitions may exist exclusively in $S'$, because $S' \supseteq S''$.

Statement 2 follows from Definition 12 of a unique minimum and the first statement. ∎

The second theorem generalizes the domination by composition to transition cost functions.

**Theorem 3.** *[Domination by Composition] Let $saturate_1$ and $saturate_2$ be transition saturators for the same transition system $\mathcal{T}$, state set $S' \subseteq S(\mathcal{T})$ and heuristic $h$. Let $saturate_{12} : \mathcal{C}(T(\mathcal{T})) \to \mathcal{C}(T(\mathcal{T}))$ be the composition of these saturators, i.e. $saturate_{12}(tcf) = saturate_2(saturate_1(tcf))$ for all transition cost functions $tcf \in \mathcal{C}(T(\mathcal{T}))$.*

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

**Proof:** Follows directly from Definition 16, where we required the output of a transition saturator to dominate its input. ∎

Theorem 2 and 3 help us in finding reasonable transition saturators from the set of possible transition saturators. Seipp and Helmert (2019) described a dominating saturator as being more "economical" because it leaves more remaining cost for the next abstraction than the saturator that it dominates. In the rest of this chapter, we define multiple transition saturators. Most of them are generalizations of the operator saturators given by Seipp and Helmert (2019).

### Transition Saturators

In the definition of each transition saturator, we consider a weighted concrete transition system $\langle \mathcal{T}, tcf \rangle$ where $tcf$ describes the current remaining transition cost function and an abstraction heuristic $h$ for $\langle \mathcal{T}, tcf \rangle$ with underlying weighted abstract transition system $\langle \mathcal{T}', tcf' \rangle$.

**all$_t$** The *all$_t$* transition saturator preserves the heuristic values of all states and is the one that was considered previously by Keller et al. (2016a). It computes the unique minimum saturated transition cost function $mstcf$ by setting the consistency constraint $h(tcf, s) \leq h(tcf, s') + mstcf(t)$ tight for all transitions $t = s \xrightarrow{l} s' \in T(\mathcal{T})$. This can be enforced by setting

$$
mstcf(t) = \begin{cases} h(tcf, s) - h(tcf, s'), \\ \qquad \text{if } h(tcf, s') < \infty \\ -\infty, \text{ otherwise} \end{cases}
$$

In the case of a nonnegative transition saturator $all_t^+$, replace all negative saturated transition costs with 0. In the case of a general transition saturator *all$_t$*, we have to handle infinite and negative costs in the input with the three preprocessing steps described by Seipp and Helmert (2019) as follows.

Firstly, for every transition $t$ with current remaining cost $tcf(t) = \infty$, we set $mstcf(t) = \infty$. It may assign a higher cost to $t$ than necessary, and the resulting saturated transition might not be minimal. This is not an issue regarding the minimality of the cost function because the remaining cost of $\infty$ will remain $\infty$ forever, independently of what we assign to $mstcf(t)$ because $\infty - \infty = \infty$ under left addition. The necessity to use $mstcf(t) = \infty$ comes from the fact that a transition with weight $\infty$ carries the information of a "blocking" transition according to the path addition rules. A transition becomes a "blocking" transition if a previous abstraction detects that the transition is part of the state space with infinite heuristic values.

Secondly, for every transition $t$ with current remaining cost $tcf(t) < \infty$ that starts in a state with a heuristic value of $-\infty$, we set $mstcf(t) = -\infty$ because the heuristic value is preserved by any saturated transition cost that is upper bounded by the input transition cost. Therefore $mstcf(t) = -\infty$ is an optimal choice because it leaves the most remaining transition cost for later abstractions.

Third, for every transition $t$ with current remaining cost $tcf(t) < \infty$ that ends in a state with a heuristic value of $\infty$, we set $mstcf(t) = -\infty$ because the heuristic value is $\infty$ independently of the saturated cost for this transition.

In the case of an NNG transition saturator, we additionally provide a modified version $all'_t$ where we replace saturated transition costs of $-\infty$ with 0. As a result, less remaining costs are available in later abstractions but using a saturated cost of 0 requires no computational effort in states with infinite heuristic value. The drawback is that later abstractions might not "block" transitions on abstract goal paths (as described in the first preprocessing step).

**spd$_t$** The $spd_t$ transition saturator is a nonnegative transition saturator and preserves the heuristic values of all states. Its name is derived from the Shortest Path Discovery (SPD) problem (Szepesvári 2004) where we are given a transition system $\mathcal{T}'$, a function $query : T(\mathcal{T}') \to \mathbb{R}_0^+ \cup \{\infty\}$ that returns the cost of a transition. The objective is to find the goal distance for every state $s \in S(\mathcal{T}')$, such that the number of evaluations of the function $query$ is as small as possible.

The SPD problem occurs in the Saturated Transition Cost Partitioning algorithm as follows: In each iteration, the algorithm computes the abstraction heuristic given the next abstraction in sequence and the remaining transition cost function. The remaining transition cost function is available in the form of a data structure based on decision diagrams (Keller et al. 2016a; Speck, Geißer, and Mattmüller 2018a; Geißer, Keller, and Mattmüller 2016). Definition 9 shows how to compute the abstract transition cost function from the given transition cost function. Our experiments have shown that this is a significant performance bottleneck. Thus, lowering the number of computations of abstract transition costs has the potential for improvements.

We use a modification of Dijkstra's algorithm that is a greedy SPD algorithm. The modification assigns the smallest remaining cost of 0 to the abstract transition (and every concrete transition that induces it) if the goal distance of the source state of the currently evaluated abstract transition is already known (Szepesvári 2004).

**reach$_t$** The $reach_t$ transition saturator is a generalization of the $reach_o$ operator saturator (Seipp and Helmert 2019). It preserves the heuristic values of all states that are reachable in a forward search. The preimage $S'$ of all states that are reachable in the abstract transition system overapproximates the set of reachable states. For every unreachable state $s \in S(\mathcal{T}) \setminus S'$ we set the heuristic estimate to $h(tcf, s) = -\infty$ because they are never visited in a forward search (Seipp and Helmert 2019).

In the case of a nonnegative transition saturator $reach_t^+$, apply the nonnegative transition saturator $all_t^+$ on the modified heuristic values. In the case of a general transition saturator, apply the transition saturator $all_t$ afterwards. In the case of an NNG transition saturator, we additionally provide a modified version $reach'_t$ that applies the $all'_t$ transition saturator instead.

**perim$_t$** The $perim_t$ transition saturator is a generalization of the $perim_o$ operator saturator (Seipp and Helmert 2019) and preserves the heuristic values of all states that are within a perimeter of $k$ to a goal. The idea is that it is more important to preserve heuristic values of states that are closer to a goal (e.g., Holte et al., 2004; Torralba, Linares López, and Borrajo, 2018). The set of states within perimeter $k > 0$ is $S_k = \{s \in S(\mathcal{T}) \mid h(tcf, s) \leq k\}$.

In our experiments, we set the heuristic value to $k$ for all states outside of $S_k$ and apply the nonnegative $all_t^+$ transition saturator on the modified heuristic values. This allows efficient computation of a saturated transition cost function with the guarantee that the heuristic does not contain negative cost cycles in reachable states (Seipp and Helmert 2019). We obtain a general transition saturator with an additional composition with the general transition saturator $all_t$.

**lp$_t$** The $lp_t$ transition saturator is a generalization of the $lp_o$ operator saturator (Seipp and Helmert 2019) and preserves the heuristic value of only a single state $s \in S(\mathcal{T})$. The set of possible saturated transition cost functions to pick from does not have a unique minimum. We use an adapted version of the linear programming formulation of the $lp_o$ operator saturator to choose from the set of possible saturated transition cost functions. The LP uses the same LP-trick to encode the heuristic estimates from the saturated transition cost function. We do not consider abstract transitions between abstract states with infinite heuristic value because we can immediately determine their saturated costs as described in the $all_t$ preprocessing (Seipp and Helmert 2019).

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

$$H_a \leq C_t + H_b \quad \text{for all } a \xrightarrow{l} b = t \in T(\mathcal{T}') \tag{2}$$

$$C_t \leq tcf'(t) \quad \text{for all } t \in T(\mathcal{T}') \tag{3}$$

$$C_t \leq C_l \qquad \text{for all } a \xrightarrow{l} b = t \in T(\mathcal{T}') \tag{4}$$

$$H_{\alpha(s)} = h(s) \tag{5}$$

The variables $C_t$ encode the saturated transition cost function, the variables $C_l$ encode the saturated operator cost function, and the variables $H_a$ encode the heuristic values $h^*_{\mathcal{T}'}$. We also choose to minimize the objective of minimizing the sum of used operator costs $\min \sum_{l \in L(\mathcal{T})} C_l$. In the case of an NNG transition saturator, we additionally provide a modified version $lp'_t$ that, similar to the other primed saturators, replaces saturated transition costs of $-\infty$ with $0$.

In our experiments, we preserve the heuristic value of the initial state.

## Experiments

We implemented all transition saturators into the Fast Downward planning system (Helmert 2006). Similar to modern symbolic search planners, we utilize decision diagrams for compact representation and computation of sets of states (Kissmann and Edelkamp 2014; Torralba et al. 2014; Speck, Geißer, and Mattmüller 2018b). More specifically, we use the CUDD library (Somenzi 1994) for Binary Decision Diagrams to represent and compute transition cost functions. We conducted experiments with the Downward Lab toolkit (Seipp et al. 2017) on Intel Xeon E5-2650v2 2.60GHz processors with 64GB DDR3 1866MHz ECC registered memory. Each task was limited to a single core with 4GB of memory and a time limit of 30 minutes. We consider the same set of abstraction as the initial work on cost saturation with operator saturators (Seipp and Helmert 2019) that consists of all CEGAR abstractions with the goal and landmark diversification technique (Seipp and Helmert 2014), the systematic pattern databases (Pommerening, Röger, and Helmert 2013) and the pattern databases found by hill-climbing (Haslum et al. 2007). We order the abstractions using the static greedy order with the scoring function $\frac{h}{stolen}$ (Seipp 2017), the best ordering method considered in previous work about operator saturators (Seipp and Helmert 2019). Our evaluation only considers cost partitionings for the single order optimized for the initial state. We optimize the order independent of the transition saturator for a fair comparison. All benchmarks, code and experimental data have been published online[1].

We write $saturate_1$,$saturate_2$ for the composition $saturate_{12}$ of two saturators where the output of $saturate_1$ is applied to the input of $saturate_2$.

With $saturate_1$+$saturate_2$ we denote that $saturate_2$ is an additional run of cost saturation on the remaining cost after computing a cost partitioning with $saturate_1$. The saturated transition cost functions of both runs form a cost partitioning. Hence, summing up the heuristic estimates of both runs yields an admissible heuristic. Postponing additional runs of cost saturation makes sense if there are potentially unused costs available.

**Comparison of the spd transition saturator** In our first experiment, we compare the $all_t$ transition saturator against the $spd_t$,$\mathrm{all}_t$ transition saturator. Figure 1 shows that the number of computations of abstract transition weights from the given remaining transition cost function is up to 90 times lower. The average over all instances is 4.3 times lower. As a result, our following experiments always apply the $spd_t$ transition saturator first because it allows for faster computation of the abstract transition cost function while preserving the heuristic estimates of all states.

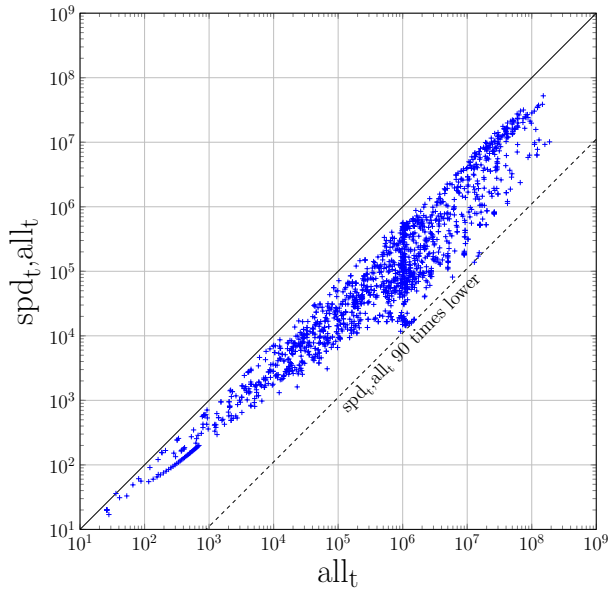


Figure 1: Comparison of the transition saturator $all_t$ against $spd_t$,$\mathrm{all}_t$ with respect to the number of computations of abstract transition weights.

**Comparison with operator saturator** In our second experiment, we compare $perim_o$,$all_o$+$all_o$ operator saturator[2] that is part of a current state-of-the-art heuristic (Seipp and Helmert 2019), against the transition saturator $perim_t$,$all'_t$+$all'_t$. The $perim_t$,$all'_t$+$all'_t$ transition saturator is the generalized counterpart to the $perim_o$,$all_o$+$all_o$ operator saturator.

Figure 2 shows the comparison with respect to the number of node expansions until the last $f$-layer. The transition saturator expands less states in 437 instances, and the opposite holds in 59 instances. Additionally, the transition saturator finds perfect heuristic estimates in significantly more instances.

In Figure 3, we compare the total time in seconds to find an optimal plan for the same saturators. The operator saturator is faster in 898 tasks where both algorithms find an optimal plan. The opposite is true in 100 cases. We conclude that Subset-Saturated Operator Cost Partitioning heavily outperforms Subset-Saturated Transition Cost Partitioning in our setup.

[1] URL to the code and data.

[2] The $perim_o$,$all_o$+$all_o$ operator saturator is denoted by perim+all in the original work. We consider *perim* only as a non-negative saturator and let composition with additional saturators decide about generality or nonnegativity.

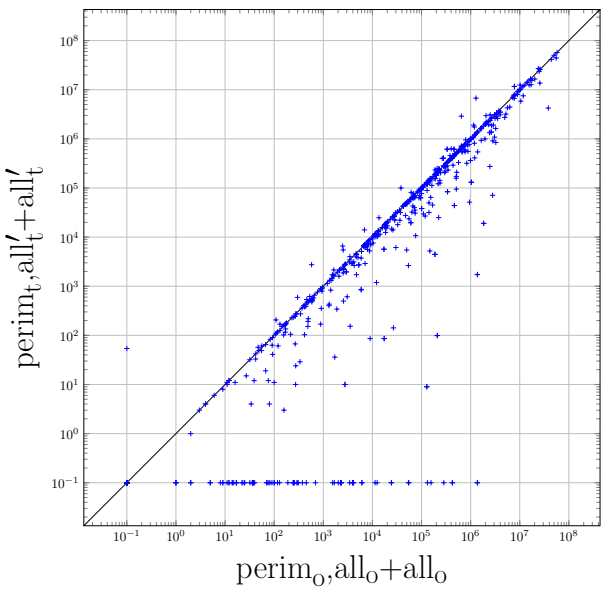


Figure 2: Comparison of the *perim$_o$,all$_o$+all$_o$* operator saturator against the *perim$_t$,all$'_t$+all$'_t$* transition saturator with respect to the number of node expansions until the last $f$-layer.

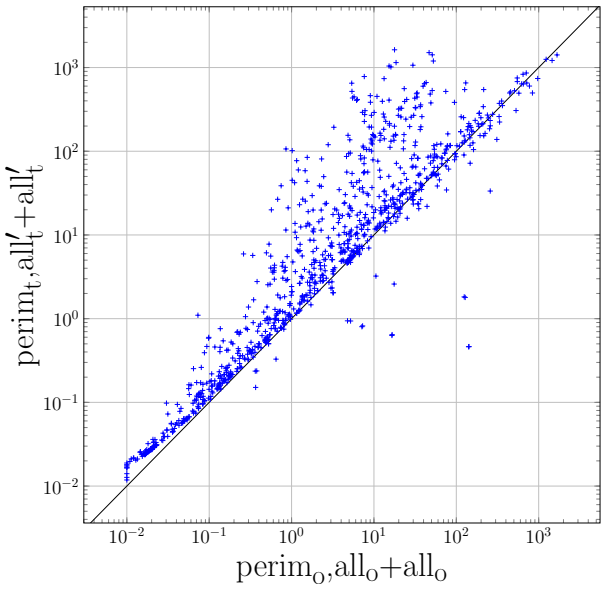


Figure 3: Comparison of the *perim$_o$,all$_o$+all$_o$* operator saturator against the *perim$_t$,all$'_t$+all$'_t$* transition saturator with respect to the total time in seconds to find an optimal plan.

**Comparison of saturators** Table 1 shows the pairwise comparison of the initial heuristic estimate for a set of reasonable saturators.

The best previous transition saturator *all$_t$* gives better heuristic estimates for the initial state than its operator saturator counterpart *all$_o$* in 332 tasks and worse in 155 tasks. Our new best transition saturator *perim$_t$,all$'_t$+all$'_t$* gives better heuristic estimates for the initial state than *all$_o$* in 623 tasks and worse in only 45 tasks.

The primed transition saturators *all$'_t$* gives worse heuristic estimates than its unprimed counterpart *all$_t$* in only 4 instances but solved 17 more instances. We obtained similar results for the comparison of other transition saturators with its primed versions. In any comparison, the total time to compute a Transition Cost Partitioning significantly improves when using the primed versions.

In our experiments, compositions that include the *perim$_t$* transition saturator have shown better results than compositions that do not use it. Postponing the *lp$_t$* transition saturator is often useless, because a leading transition saturator does not leave any slack in the consistency constraint, e.g. *reach$_t$*, *lp$_t$* returns the same heuristic as *reach$_t$* without *lp$_t$*.

| | all$_o$ | perim$_o$,all$_o$+all$_o$ | all$_t$ | all$'_t$ | reach$'_t$ | lp$'_t$ | perim$_t$,all$'_t$+all$'_t$ |
|---|---|---|---|---|---|---|---|
| all$_o$ | - | 53 | 155 | 156 | 155 | 92 | 47 |
| perim$_o$,all$_o$+all$_o$ | **476** | - | **364** | **391** | **385** | 123 | 45 |
| all$_t$ | **332** | 225 | - | **4** | 17 | 92 | 32 |
| all$'_t$ | **349** | 232 | 0 | - | 13 | 90 | 32 |
| reach$'_t$ | **387** | 244 | **47** | **66** | - | 93 | 42 |
| lp$'_t$ | **410** | **238** | **296** | **323** | **323** | - | 93 |
| perim$_t$,all$'_t$+all$'_t$ | **623** | **361** | **430** | **458** | **458** | **167** | - |

Table 1: Per-task comparison of the initial $h$-value for a subset of all saturator compositions. In every pairwise comparison we consider the tasks for which both transition saturators finished computing the initial heuristic estimate. The entry in row $r$ and column $c$ indicates the number of tasks where $r$ returns a transition cost partitioning with a better initial heuristic value than $c$. Boldface is used to indicate the winner in the pairwise comparison $(r, c)$ and $(c, r)$.

## Future work

Improving worst-case performance by improving the storage of transition cost function can greatly impact performance. Keller et al. (2016a) already suggested to use variable reordering techniques on decision diagrams which still remains untouched and open for future work.

Subset-Saturated Transition Cost Partitioning is not effective on large sets of abstractions, especially abstractions with large similarities. Considering different sets of abstractions with sufficiently large diversity could further prune unnecessary overhead.

Subset-Saturated Transition Cost Partitioning is compatible with Subset-Saturated Operator Cost Partitioning, in the

sense that we can interleave both algorithms to distinguish the context of applied actions only where it seems most promising. For example, it makes no sense to distinguish the context of an action that only induces state-changing transitions in at most one abstraction. Actions that induce many state-changing transitions in many abstractions are more likely to improve the heuristic estimates but can also negatively influence the performance tradeoff. We think that identifying these interactions can lead to improvements.

## Conclusion

We applied cost saturation for subsets to the Saturated Transition Cost Partitioning algorithm. We came up with an additional saturator that improves computing heuristic estimates of abstractions from a given transition cost function. Subset-Saturated Transition Cost Partitioning adds a significant amount of computational overhead because of taking the context of applied actions into account. Our empirical evaluation shows that this often not pays off regarding the runtime but leads to more informed heuristics.

## References


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

Geißer, F.; Keller, T.; and Mattmüller, R. 2016. Abstractions for planning with state-dependent action costs. In *Proc. ICAPS 2016*, 140–148.

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.

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.

2014. *IPC-8 planner abstracts*.

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

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

Keller, T.; Pommerening, F.; Seipp, J.; Geißer, F.; and Mattmüller, R. 2016b. State-dependent cost partitionings for cartesian abstractions in classical planning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, 3161–3169. AAAI Press.

Kissmann, P., and Edelkamp, S. 2014. Gamer and dynamic-gamer – symbolic search at ipc 2014. In *IPC-8 planner abstracts* (2014), 77–84.

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.

Pommerening, F. 2017. *New Perspectives on Cost Partitioning for Optimal Classical Planning*. Ph.D. Dissertation, University of Basel.

Seipp, J., and Helmert, M. 2013a. Additive counterexample-guided Cartesian abstraction refinement. In *Proc. AAAI 2013 Late-Breaking Papers*, 119–121.

Seipp, J., and Helmert, M. 2013b. Counterexample-guided Cartesian abstraction refinement. In *Proc. ICAPS 2013*, 347–351.

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

Seipp, J., and Helmert, M. 2019. Subset-saturated cost partitioning for optimal classical planning. In *Proc. ICAPS 2019*, 391–400.

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. 2017. A comparison of cost partitioning algorithms for optimal classical planning. In *Proc. ICAPS 2017*, 259–268.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated cost partitioning for optimal classical planning. *JAIR* 67:129–167.

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

Somenzi, F. 1994. Cudd: colorado university decision diagram package - release 3.0.0.

Speck, D.; Geißer, F.; and Mattmüller, R. 2018a. Symbolic planning with edge-valued multi-valued decision diagrams. In *Proc. ICAPS 2018*, 250–258.

Speck, D.; Geißer, F.; and Mattmüller, R. 2018b. SYMPLE: Symbolic Planning based on EVMDDs. In *IPC-9 planner abstracts*, 91–94.

Szepesvári, C. 2004. Shortest path discovery problems: A framework, algorithms and experimental results. In *Proc. AAAI 2004*, 550–555.

Torralba, Á.; Alcázar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. SymBA*: A symbolic bidirectional A* planner. In *IPC-8 planner abstracts* (2014), 105–109.

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