Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)

Subset-Saturated Transition Cost Partitioning

Dominik Drexler,[1,2] **Jendrik Seipp,**[1,3] **David Speck**[2]
[1]Linköping University, Linköping, Sweden
[2]University of Freiburg, Freiburg, Germany
[3]University of Basel, Basel, Switzerland
{dominik.drexler, jendrik.seipp}@liu.se, speckd@informatik.uni-freiburg.de

Abstract

Cost partitioning admissibly combines the information from multiple heuristics for optimal state-space search. One of the strongest cost partitioning algorithms is saturated cost partitioning. It considers the heuristics in sequence and assigns to each heuristic the minimal fraction of the remaining costs that are needed for preserving all heuristic estimates. Saturated cost partitioning has recently been generalized in two directions: first, by allowing to use different costs for the transitions induced by the same operator, and second, by preserving the heuristic estimates for only a subset of states. In this work, we unify these two generalizations and show that the resulting subset-saturated transition cost partitioning algorithm usually yields stronger heuristics than the two generalizations by themselves.

generalization →

↓ generalization	all states	subset of states
operators	(a) saturated operator CP	(c) subset-saturated operator CP
transitions	(b) saturated transition CP	(d) **subset-saturated transition CP**

Table 1: Saturated operator cost partitioning and its generalizations from operators to transitions and from all states to a subset of states.

Introduction

A* search with an admissible heuristic is one of the main techniques to solve planning tasks optimally (Hart, Nilsson, and Raphael 1968; Pearl 1984). Since a single heuristic is usually unable to capture enough information about the task, we often need to combine the information from multiple heuristics admissibly. The preferable way of doing so is cost partitioning (CP, Katz and Domshlak 2008; Keller et al. 2016). A transition cost partitioning for a weighted transition system distributes the cost of each transition over multiple heuristics such that the sum of costs for each transition does not exceed the original transition cost. If each component heuristic is admissible, then the overall cost-partitioned heuristic, i.e., the sum of the component heuristics evaluated under the partitioned cost functions, is also admissible. Many cost partitioning papers from the literature consider an important special case of transition cost partitioning: operator cost partitioning (e.g., Haslum, Bonet, and Geffner 2005; Haslum et al. 2007; Katz and Domshlak 2008, 2010; Pommerening, Röger, and Helmert 2013). In an operator cost partitioning, all transitions that are induced by the same operator have the same cost.

One of the strongest approaches to compute cost-partitioned heuristics is saturated cost partitioning (Seipp, Keller, and Helmert 2020). Saturated cost partitioning considers the heuristics in sequence and assigns the first heuristic the minimal fraction of the remaining costs that it needs to preserve its heuristic estimates. Then the algorithm uses the remaining costs to treat the subsequent heuristics in the same way. In its original form (Table 1a), the saturated cost partitioning algorithm computes an operator cost partitioning and always preserves the heuristic estimates for all states under the remaining cost function (Seipp and Helmert 2014).

A generalization of saturated operator cost partitioning is saturated transition cost partitioning (Table 1b) where costs are assigned directly to transitions (Keller et al. 2016). Transition cost partitioning is more general than operator cost partitioning because each operator can induce multiple transitions. While there is no dominance relation between the operator and transition version of saturated cost partitioning, Keller et al. (2016) showed empirically that the transition version often yields more accurate heuristics than the operator version. However, our experiments show that the computational overhead caused by allowing more expressive cost assignments is often too large for the additional heuristic accuracy to lead to solving more tasks.

More recently, Seipp and Helmert (2019) generalized saturated operator cost partitioning in a different direction. They allow the saturated cost partitioning algorithm to preserve the heuristic estimates of only a subset of states (Table 1c). Their empirical evaluation shows that this approach yields more accurate heuristics and achieves state-of-the-art results on the benchmark set of the International Planning Competition (IPC).

To unify these two generalizations, we introduce subset-saturated transition cost partitioning (Table 1d). It assigns costs to individual transitions and preserves the heuristic estimates of only a subset of states. We show that subset-saturated transition cost partitioning preserves admissibility and that it usually yields stronger heuristics than the two generalizations alone.

Background

In this section, we define the concepts that form the foundation of subset-saturated transition cost partitioning. We consider SAS$^+$ *planning tasks* (Bäckström and Nebel 1995) with operator costs. A planning task consists of a set of finite-domain variables, a finite set of operators, an initial state, a goal condition and an operator cost function. The details of planning tasks are unimportant for our contributions, but the important part is that a planning task induces a transition system.

Transition Systems

A *transition system* describes the dynamics of a state-based system and is also called a state space. For all our definitions we follow the notation used by Seipp and Helmert (2019).

Definition 1 (Transition System)**.** *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 $\langle s, l, s' \rangle$ with $s, s' \in S(\mathcal{T})$ and $l \in L(\mathcal{T})$, an initial state $s_I(\mathcal{T}) \in S(\mathcal{T})$, and a set of goal states $S_\star(\mathcal{T}) \subseteq S(\mathcal{T})$.*

A *path* from $s_0 \in S(\mathcal{T})$ to $s_n \in S(\mathcal{T})$ is a sequence $\pi = \langle s_0, l_1, s_1, \ldots, s_{n-1}, l_n, s_n \rangle$ where $\langle s_{i-1}, l_i, s_i \rangle \in T(\mathcal{T})$ for all $i = 1, \ldots, n$. It is called a *goal path* if it ends in a goal state $s_\star \in S_\star(\mathcal{T})$. The objective of state-space search is to find a goal path from the initial state $s_I(\mathcal{T})$.

Definition 2 (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 $tcf : T(\mathcal{T}) \to \mathbb{R} \cup \{-\infty, \infty\}$. 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 $\mathcal{C}(\mathcal{T})$ for the set of all transition cost functions for $\mathcal{T}$ and $\mathcal{C}_{\geq 0}(\mathcal{T})$ 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 where each transition with the same label is assigned the same cost value. Therefore, an operator cost function is a mapping from (operator) labels to costs. When we combine a transition system $\mathcal{T}$ with a transition cost function $tcf \in \mathcal{C}(\mathcal{T})$, we obtain a *weighted transition system* $\langle \mathcal{T}, tcf \rangle$.

We use *left-addition* and *path-addition* (Seipp and Helmert 2019) to handle arithmetic expressions that contain cost values of $+\infty$ and $-\infty$. The symbols $+$ (infix) and $\sum$ (prefix) denote the *left-addition* operation. Left-addition over finite values is the usual addition. Expressions that contain infinities are defined as $\infty + x = \infty$, $-\infty + x = -\infty$ for all x, including x being ∞ or $-\infty$, $x + \infty = \infty$ for all $x \neq -\infty$, and $x + (-\infty) = -\infty$ for all $x \neq \infty$. This operation is associative but not commutative. Intuitively, a left-addition that contains mixed infinities evaluates to the leftmost infinity. In cost partitioning, we use left-addition for summing up multiple heuristic values and transition costs.

The symbols $\oplus$ (infix) and $\bigoplus$ (prefix) denote the *path-addition* operation. Path-addition over finite values is the usual addition. Expressions that contain infinities are defined as $x \oplus y = \infty$ if $x = \infty$ or $y = \infty$ and $x \oplus (-\infty) = -\infty$ for all $x \neq \infty$. This operation is associative and commutative. Intuitively, a path-addition evaluates to $+\infty$ if it contains at least one $+\infty$. We use path-addition to define the cost of a path in a transition system. The cost of a path $\pi = \langle s_0, l_1, s_1, \ldots, s_{n-1}, l_n, s_n \rangle$ with $t_i = \langle s_{i-1}, l_i, s_i \rangle \in T(\mathcal{T})$ for all $i = 1, \ldots, n$ in a weighted transition system $\langle \mathcal{T}, tcf \rangle$ 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 ∞ is non-existent.

The *goal distance* $h^*_{\mathcal{T}}(tcf, s)$ 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. Notice that a goal distance $h^*_{\mathcal{T}}(tcf, s) = -\infty$ can result from infinitely cheap labels but also from finite negative-cost cycles. A goal path π from s is optimal under the given transition cost function tcf if $cost(tcf, \pi) = h^*_{\mathcal{T}}(tcf, s)$. In optimal classical planning, we are interested in finding an optimal goal path from the initial state or proving that no such goal path exists.

Heuristic Search

A *heuristic* is a function that estimates the goal distance of a state under a given cost function (Pearl 1984).

Definition 3 (Heuristic)**.** *Let $\langle \mathcal{T}, tcf \rangle$ be a weighted transition system. A heuristic $h(tcf, s) \to \mathbb{R} \cup \{-\infty, \infty\}$ for $\mathcal{T}$ maps each state $s \in S(\mathcal{T})$ to a goal distance estimate under the transition cost function tcf. The heuristic h is admissible if $h(tcf, s) \leq h^*_{\mathcal{T}}(tcf, s)$ for all states $s \in S(\mathcal{T})$.*

The A^* search algorithm with an admissible heuristic is guaranteed to find optimal goal paths (Hart, Nilsson, and Raphael 1968).

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 4 (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 α, β if (1) $\alpha(s_I(\mathcal{T})) = s_I(\mathcal{T}')$, (2) $\alpha(s_\star) \in S_\star(\mathcal{T}')$ for all $s_\star \in S_\star(\mathcal{T})$, and (3) $\langle \alpha(s), l, \alpha(s') \rangle \in T(\mathcal{T}')$ and $\beta(\langle s, l, s' \rangle) = \langle \alpha(s), l, \alpha(s') \rangle$ for all $\langle s, l, s' \rangle \in T(\mathcal{T})$.*

We refer to $\mathcal{T}$ as the concrete transition system and $\mathcal{T}'$ as the abstract transition system. An abstraction heuristic is a function that maps each concrete state to the goal distance of its corresponding abstract state in the abstraction.

Definition 5 (Abstraction heuristic)**.** *Let $\langle \mathcal{T}, tcf \rangle$ be a weighted transition system and $\mathcal{T}'$ be an abstraction of $\mathcal{T}$ with abstraction mappings α, β.*

The abstract transition cost function $tcf' \in \mathcal{C}(\mathcal{T}')$ defines the cost of each abstract transition $t' \in T(\mathcal{T}')$ as:

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

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 representation size of an abstraction heuristic depends on the representation size of the abstraction mappings α, β. We consider Cartesian abstractions (Ball, Podelski, and Rajamani 2001) where the set of all concrete states that map to the same abstract state forms a Cartesian set. A Cartesian set of states has the form $D_1 \times \ldots \times D_n$ where each D_i is a subset of the domain of the i-th variable of the planning task. Furthermore, the property of Cartesianness is closed under operations such as intersection and operator regression (Seipp and Helmert 2013). Operator regression allows deriving the abstraction mapping β from the factored representation of the operators of the planning task and the abstraction mapping α (Keller et al. 2016).

Abstraction heuristics under general transition cost functions are admissible (Drexler, Seipp, and Speck 2021). Intuitively, this is the case because every goal path in the concrete transition system corresponds to a goal path in the abstract transition system and the minimization in the abstract transition cost function ensures that the abstraction heuristic does not overestimate any goal distance.

Transition Cost Partitioning

A *transition cost partitioning* splits a given transition cost function into a sequence of transition cost functions such that the sum is bounded from above by the original transition cost function (Keller et al. 2016).[1]

Definition 6 (Transition Cost Partitioning)**.** *A transition cost partitioning for a weighted transition system $\langle \mathcal{T}, tcf \rangle$ is a tuple $\langle tcf_1, \ldots, tcf_n \rangle \in \mathcal{C}(\mathcal{T})^n$ whose sum is bounded from above by tcf, i.e., $\sum_{i=1}^{n} tcf_i(t) \leq tcf(t)$ for all $t \in T(\mathcal{T})$.*

The following proposition generalizes the results from previous work to general transition cost functions and shows that a transition cost partitioning induces an admissible heuristic (Katz and Domshlak 2008; Pommerening et al. 2015; Keller et al. 2016; Seipp, Keller, and Helmert 2020).

Proposition 1 (Admissibility; proof in Drexler, Seipp, and Speck 2021)**.** *Let $\langle tcf_1, \ldots, tcf_n \rangle$ be a transition cost partitioning for a weighted transition system $\langle \mathcal{T}, tcf \rangle$ and let $\langle h_1, \ldots, h_n \rangle$ be admissible heuristics for $\mathcal{T}$. Then $h(s) := \sum_{i=1}^{n} h_i(tcf_i, s)$ is an admissible heuristic. We say that h is a cost-partitioned heuristic.* □

[1] In contrast to Keller et al. (2016), we allow transition cost functions to map to $-\infty$ and set $tcf(t) = -\infty$ if we detect that transition t can never be part of a goal path.

An optimal transition cost partitioning over a set of heuristics is a transition cost partitioning that provides the highest heuristic estimate for a given state. We can compute an optimal transition cost partitioning by compiling the input planning task into a task where each operator corresponds to a single transition of the input task, followed by computing the optimal operator cost partitioning (Katz and Domshlak 2008) for the compiled task. The result can be mapped to an optimal transition cost partitioning but the computation is not feasible in practice due to the exponential blowup in the compilation (Keller et al. 2016). Instead, we use a greedy method called saturated cost partitioning (Seipp, Keller, and Helmert 2020).

Subset-Saturated Transition Cost Partitioning

Saturated cost partitioning (SCP) considers the heuristics in sequence. Beginning with the first heuristic in the sequence, the algorithm computes the minimum cost function under which the heuristic still yields the same estimates, i.e., the *saturated cost function*. Then it subtracts this cost function from the original cost function and uses the resulting costs, i.e., the *remaining cost function*, to treat subsequent heuristics in the same way. The set of computed saturated cost functions forms a cost partitioning.

In this section, we define subset-saturated transition cost partitioning where the saturated cost function is a transition cost function (Keller et al. 2016), and it suffices to preserve the heuristic estimates of a subset of states (Seipp and Helmert 2019). Before we formally define saturated transition cost functions, we introduce the notion of a dominating transition cost function. We say that a transition cost function tcf *dominates* a transition cost function tcf' defined on the same transition system $\mathcal{T}$, i.e., $tcf \leq tcf'$, if $tcf(t) \leq tcf'(t)$ for all transitions $t \in T(\mathcal{T})$. Furthermore, the transition cost function tcf is the unique minimum of a set of transition cost functions TCF iff tcf dominates each transition cost function $tcf' \in TCF$.

Definition 7 (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 $stcf \in \mathcal{C}(\mathcal{T})$ is saturated for S', h and tcf if*

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

Property 1 of a saturated transition cost function ensures that only a fraction of the remaining transition cost function tcf is used, while Property 2 ensures that the heuristic estimates of a subset of states S' are preserved in each SCP iteration. A saturated transition cost function always exists because tcf itself is a saturated transition cost function. We formalize the computation of a saturated transition cost function with a *transition saturator*. A transition saturator is a function that takes as an input a heuristic h, a subset of states S', and a transition cost function tcf and outputs a saturated transition cost function for h, S', and tcf.

Definition 8 (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*

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

In contrast, an operator saturator is a transition saturator that only allows for operator cost functions in the input and output (Seipp and Helmert 2019). We parameterize saturated transition cost partitioning with transition saturators to allow saturation for a subset of states.

Definition 9 (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 admissible 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 a transition saturator for* $S' \subseteq S(\mathcal{T})$ *and* h_i *for all* $1 \leq i \leq n$*. The saturated transition cost partitioning* $\langle tcf_1, \ldots, tcf_n \rangle$ *of the transition cost function* tcf *induced by* $Saturate$ *is defined as:*

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

The subtraction in the definition of $remain_i$ is defined in terms of left-addition, i.e., $a - b := a + (-b)$. Using the same reasoning as Seipp and Helmert (2019) it is easy to see that saturated transition cost partitionings obtained with Definition 9 are indeed transition cost partitionings.

Subset-saturated transition cost partitioning has three major choice points that influence the accuracy of the cost-partitioned heuristic. These are the set of heuristics, the order of the heuristics, and the transition saturators. In this work, we focus on the transition saturators.

Due to the greediness of the SCP algorithm, it is not guaranteed that we obtain a more accurate cost-partitioned heuristic if we use fewer costs in an SCP iteration to preserve the same heuristic estimates. This is also the reason why all four SCP variants from Table 1 are pairwise incomparable theoretically (Keller et al. 2016; Seipp and Helmert 2019). However, Seipp and Helmert (2019) showed that using costs more *economically* usually yields stronger heuristics.

In the following, we present three ways to obtain more economical transition saturators. First, we obtain a more economical transition saturator by preserving the heuristic estimates of a subset of states (Seipp and Helmert 2019). Consider a heuristic h for a weighted transition system $\langle \mathcal{T}, tcf \rangle$, and $S'' \subseteq S' \subseteq S(\mathcal{T})$. Definition 7 shows that every saturated transition cost function for h, S' and tcf is also a saturated transition cost function for h, S'' and tcf because the latter has fewer constraints on the saturated transition cost function than the former.

Second, we obtain a more economical transition saturator by composition (Seipp and Helmert 2019). Consider two transition saturators $saturate_1, saturate_2$ for a subset of states $S' \subseteq S(\mathcal{T})$. Then the composition $saturate_{12}(tcf) := saturate_2(saturate_1(tcf))$ dominates $saturate_1(tcf)$ for all transition cost functions $tcf \in \mathcal{C}(\mathcal{T})$. The result holds because according to Definition 8, the output of a transition saturator always dominates its input.

Third, we obtain a more economical transition saturator by taking an existing operator saturator and allowing to use transition cost functions in the input and output. This can be more economical because transition cost functions are a generalization of operator cost functions.

In the following section, we define transition saturators that use these three observations.

Transition Saturators

We introduce five transition saturators, four of which are generalizations of operator saturators (Seipp and Helmert 2019).

Saturate for All States (all_t)

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

$$mstcf(t) = h(tcf, s) \ominus h(tcf, s').$$

The operator $\ominus$ behaves like regular subtraction in the finite case and handles infinities as $x \ominus y = -\infty$ iff $x = -\infty$ or $y = \infty$ and $x \ominus y = \infty$ iff $x = \infty \neq y$ or $x \neq -\infty = y$. With the results by Seipp, Keller, and Helmert (2020) it follows immediately that $mstcf$ is the unique minimum saturated transition cost function.

Saturate for All States Faster (fast_t)

The transition saturator $fast_t$ also preserves the heuristic estimates of all states. It is a novel transition saturator and tailored at improving the efficiency of computing all goal distances in an abstraction under a given transition cost function (Definition 5).

Consider an abstraction heuristic h for a weighted transition system $\langle \mathcal{T}, tcf \rangle$ with underlying abstraction $\mathcal{T}'$ of $\mathcal{T}$. A straightforward way to compute all goal distances in $\mathcal{T}'$ is: (1) compute the abstract transition cost function tcf', and (2) compute $h^*_{\mathcal{T}'}(tcf', s)$ for all $s \in S(\mathcal{T}')$ using a backwards exploration from the goal states with an algorithm that can handle general cost functions. Exploratory experiments showed that the minimization in step 1 (see Definition 5) is a performance bottleneck. Fortunately, we can speed up the computation by interleaving both steps.

This is possible since we only need to compute goal distances under nonnegative cost functions: $remain_0$ is nonnegative since it is the cost function of the input planning task. Since Definition 7 ensures that only a fraction of $remain_i$ is allocated to each heuristic for $1 \leq i \leq n$, all subsequent $remain_i$ are nonnegative as well.

With only nonnegative cost functions we can modify Dijkstra's algorithm (1959) slightly and perform both steps simultaneously. The modified algorithm works as follows: let $t = \langle s, l, s' \rangle \in \mathcal{T}'$ be the abstract transition that Dijkstra's

algorithm expands next. Furthermore, let $d(s)$ be the current cheapest cost of a path from $s \in S(\mathcal{T}')$ to a goal. If $d(s) \leq d(s')$ then we set $tcf'(t) = 0$ because it is the cheapest goal path from s that uses t and does not make $d(s)$ any cheaper, i.e., there exists a goal path from s that does not use t and is cheaper. Otherwise we compute $tcf'(t)$ as shown in Definition 5 and update $d(s)$ if the goal path that uses t makes $d(s)$ cheaper. See Drexler, Seipp, and Speck (2021) for pseudo-code of the procedure.

The above algorithm is sound and complete for the problem of computing goal distances under the given remaining transition cost function (Drexler, Seipp, and Speck 2021). This is intuitively the case because each cost assignment is upper bounded by the remaining transition cost function and never introduces a shortcut. The above problem that we are solving is closely related to the shortest path discovery problem (Szepesvári 2004) with the additional assumptions that the transition weights are nonnegative and that we have multiple sources and multiple targets.

Saturate for Reachable States (reach_t)

The transition saturator $reach_t$ is a generalization of the operator saturator $reach_o$ (Seipp and Helmert 2019). For a given heuristic h for a weighted transition system $\langle \mathcal{T}, tcf \rangle$, this saturator preserves the heuristic estimates of all states that are reachable from a given state (the initial state in our experiments). We set the heuristic estimate of all states $s \in S(\mathcal{T})$ that are mapped to an unreachable abstract state under h to $-\infty$ and apply all_t on the modified heuristic estimates. Since $reach_t$ uses all_t, the resulting cost function is the unique minimum saturated transition cost function for the set of concrete states mapped to reachable abstract states.

Saturate for a Perimeter (perim_t)

The transition saturator $perim_t$ is a generalization of the operator saturator $perim_o$. For a given heuristic h for a weighted transition system $\langle \mathcal{T}, tcf \rangle$, $perim_t$ preserves the heuristic estimates of all states that are within a perimeter of k to a goal state (Seipp and Helmert 2019). The idea is that it is more important to preserve heuristic estimates of states that are closer to a goal state (Holte et al. 2004; Torralba, Linares López, and Borrajo 2018). The set of states within perimeter $k \geq 0$ is $S_k = \{s \in S(\mathcal{T}) \mid h(tcf, s) \leq k\}$.

To efficiently compute a saturated transition cost function for S_k and a given heuristic h, we only allow for nonnegative transition cost functions in the input (Seipp and Helmert 2019). In our experiments, we set $k = h(tcf, s)$, where tcf is the cost function that is passed to perim_t and $s \in S(\mathcal{T})$ is a state for which we aim to optimize the resulting cost-partitioned heuristic, e.g., the initial state.

Saturate for a Single State (lp_t)

The transition saturator lp_t is a generalization of the operator saturator lp_o. For a given heuristic h of a weighted transition system $\langle \mathcal{T}, tcf \rangle$ with underlying abstraction $\mathcal{T}'$ and abstract transition cost function tcf', lp_t preserves the heuristic estimate of a single state $s \in S(\mathcal{T})$ while being as economical as possible (Seipp and Helmert 2019).

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 operator saturator lp_o to choose from the set of possible saturated transition cost functions.

$$\min \sum_{l \in L(\mathcal{T})} C_l \cdot \mathbb{1}_{\{-\infty < C_l < \infty\}}$$

$$\begin{aligned}
H_{s_\star} &\leq 0 && \text{for all } s_\star \in S_\star(\mathcal{T}') \\
H_a &\leq C_t + H_b && \text{for all } \langle a, l, b \rangle = t \in T(\mathcal{T}') \\
C_t &\leq tcf'(t) && \text{for all } t \in T(\mathcal{T}') \\
H_{\alpha(s)} &= h^*_{\mathcal{T}'}(tcf', \alpha(s)) \\
C_t &\leq C_l && \text{for all } \langle a, l, b \rangle = t \in T(\mathcal{T}')
\end{aligned}$$

The variables C_t encode the abstract transition weights of the saturated transition cost function, the variables C_l encode the saturated operator cost function, and the variables H_a encode the abstract goal distance of $a \in S(\mathcal{T}')$. Note that minimizing the sum of finite saturated transition costs in the objective function is infeasible in practice because there may be exponentially many transitions in the concrete transition system. Instead, we minimize the sum of finite saturated *operator* costs. In our experiments, we preserve the heuristic estimate of the initial state $s_I(\mathcal{T})$.

In the remainder of this section, we discuss implementation details for the transition saturators. First, we discuss the concern of saturating for unsolvable or unreachable states. Second, we discuss the issue of heuristic reevaluation under general transition cost functions.

Negative Infinity in Saturator Outputs

The runtime of the subtraction of the saturated transition cost function in Definition 9 depends on the number of abstract transitions for which the saturated cost is unequal to 0. More precisely, the subtraction of the saturated costs of all (exponentially many) concrete transitions that induce the same abstract transition requires a sequence of operations in the data structure based on binary decision diagrams (Bryant 1985) that depend on the size of the data structure itself. For abstraction heuristics with significantly more unsolvable states and unreachable states this can become prohibitively expensive because each transition that ends in an unsolvable state or starts in an unreachable state has a saturated cost of $-\infty$. The benefit of saturating with $-\infty$ is to detect additional abstract paths that are not goal paths. We observed that the benefit is relatively small compared to runtime cost. Thus, we also use modified versions of the transition saturators above for the case of a nonnegative input cost functions that substitute each saturated cost of $-\infty$ by 0, making the subtraction from the remaining transition cost function trivial.

Negative Costs in Saturator Inputs

Transition saturators require the ability to reevaluate the heuristic under the given input transition cost function. The first reevaluation under $remain_i$ is generally not too costly because $remain_i$ is always nonnegative if the cost function of the planning task is nonnegative. However, if a transition saturator saturates for a subset of states S' and we later

reevaluate the heuristic for states $s \notin S'$, then this requires algorithms like Bellman-Ford (Ford 1956; Bellman 1958) if the saturated transition cost function contains negative costs. Seipp and Helmert (2019) report that a heuristic reevaluation with Bellman-Ford algorithm was prohibitively expensive in the case of operator saturators. Therefore, they propose to directly extract a lower bound on the heuristic estimates from the output of the saturators, sacrificing heuristic accuracy.

In contrast, the transition saturator of a corresponding operator saturator has the ability to tighten the consistency constraint set on each transition. In this case, a heuristic does not change its estimates after reevaluation under the saturated transition cost function and we can directly extract the heuristic from the output of the saturator without sacrificing heuristic accuracy.

Experiments

We implemented all transition saturators in the Fast Downward planning system (Helmert 2006). Similar to modern symbolic search planners, we use decision diagrams for the compact representation and computation of sets of states (Kissmann, Edelkamp, and Hoffmann 2014; Torralba et al. 2014; Speck, Geißer, and Mattmüller 2018). More specifically, we use the CUDD library (Somenzi 2015) for binary decision diagrams with the default Fast Downward variable order to represent and manipulate transition cost functions. We conduct experiments with the Downward Lab toolkit (Seipp et al. 2017) on a compute cluster with nodes equipped with two Intel Xeon Gold 6242 32-core CPUs, 20 MB cache, and 188 GB shared RAM running Ubuntu 20.04 LTS 64-bit. The benchmark set consists of all 1827 instances from the optimization track of the 1998–2018 International Planning Competitions that do not have conditional effects. Each task is limited to a single core with 6 GB of memory and a time limit of 30 minutes. We consider the same set of abstractions as in the work on operator saturators by Seipp and Helmert (2019) that consists of Cartesian abstractions of the goal and landmark decomposition methods (Seipp and Helmert 2014), systematic pattern databases of sizes 1 and 2 (Pommerening, Röger, and Helmert 2013), and the pattern databases found by hill climbing (Haslum et al. 2007). We order the heuristics greedily with the scoring function $\frac{h}{stolen}$, that measures how well a heuristic balances the two objectives of having a high heuristic estimate and stealing low costs from other heuristics (Seipp, Keller, and Helmert 2020). We always use the operator saturator all_o during the optimization of the order for a fair comparison. All benchmarks, code, and experimental data are available online (Drexler, Seipp, and Speck 2021).

Regarding saturator composition, we always use $fast_t$ first in each transition saturator composition and omit writing $fast_t$ for improved readability. For example, instead of $fast_t$,$perim_t$, we write $perim_t$. We declare exceptions to this rules with superscript $*$. For example, when using $perim_t$ without first applying $fast_t$, we write $perim_t^*$. Furthermore, we add the superscript $-\infty$ to transition saturators for which we allow saturated costs of negative infinity. If we replace them by 0, we omit the superscript.

Saturators such as $perim_t$ or lp_t often leave costs unused after the first run of subset-saturated transition cost partitioning. In this case, an additional run with all_t or $reach_t$ on the remaining costs of the first run often improves the heuristic estimates of states that the former saturators did not optimize for. The sum of both cost-partitioned heuristics is admissible because the second run uses the remaining costs after the first run. We write $saturate_1$+$saturate_2$ to denote that $saturate_2$ is an additional run on the remaining costs after the first run with $saturate_1$.

Single Order of the Heuristics

In this section, we evaluate cost-partitioned heuristics for a single order of the heuristics optimized for the initial state.

Comparison of Transition Saturators Table 2a shows the pairwise comparison of a set of transition saturators. Saturated transition cost partitioning with the transition saturator $all_t^{*,-\infty}$ as in the work by Keller et al. (2016) leads to solving 1024 tasks. Additional composition with $fast_t$ in the transition saturator $\text{all}_\text{t}^{-\infty}$ clearly pays off because the coverage increases to 1042 tasks even though the same heuristic is computed. Replacing saturated transition costs of $-\infty$ by 0 in the transition saturator all_t further increases coverage to 1066 tasks while computing worse heuristics for the initial state in only 8 tasks.

Preserving the heuristic estimates of only reachable states with $reach_t$ decreases the coverage but increases the accuracy compared to all_t. In general, preserving the heuristic estimates of only reachable states is preferable, but if an abstraction contains many abstract states and transitions, as in our case with up to 10^9 states, then even a simple run of Dijkstra's algorithm is expensive. The coverage decreases because computing the set of reachable abstract states outweighs the additional accuracy. Preserving the heuristic estimates of only the initial state, lp_t+$reach_t$ computes stronger heuristics than the previous saturators but solves only 637 tasks.

Compositions with the transition saturator $perim_t$ yield the best results. The additive composition $perim_t$+all_t has the highest coverage of 1083 tasks and wins all pairwise comparisons for the heuristic estimate of the initial state. The objective of $perim_t$ and lp_t are closely related: both preserve the heuristic estimate of the initial state but not the heuristic estimate of states that are further away from the goal. The difference in accuracy results from lp_t being too optimistic: while saturating for a heuristic h, lp_t often saves costs for subsequent heuristics, even though h would benefit from the costs and subsequent heuristics cannot use the saved costs. Overall however, the results in Table 2a show that being economical and preserving the heuristic estimates of relevant states only leads to more accurate cost-partitioned heuristics.

Comparison to Operator Saturators Table 2b shows the pairwise comparison of four operator and transition saturators. For each of the four saturated cost partitioning variants from Table 1, we choose one representative saturator: all_o

	$\text{all}_t^{*,-\infty}$	$\text{all}_t^{-\infty}$	all_t	reach_t	lp_t^*+reach_t	perim_t+all_t
$\text{all}_t^{*,-\infty}$	–	**0**	**7**	18	4	34
$\text{all}_t^{-\infty}$	**0**	–	**8**	19	4	35
all_t	0	0	–	14	0	36
reach_t	**49**	**49**	**69**	–	39	45
lp_t^*+reach_t	**115**	**115**	**123**	**123**	–	68
perim_t+all_t	**482**	**487**	**517**	**511**	**317**	–
Coverage	1024	1042	1066	1060	637	**1083**

(a) Comparison of transition saturators.

	all_o	perim_o+all_o	$\text{all}_t^{*,-\infty}$	perim_t+all_t
all_o	–	47	164	59
perim_o+all_o	**488**	–	**390**	55
$\text{all}_t^{*,-\infty}$	**345**	236	–	34
perim_t+all_t	**683**	**400**	**482**	–
Coverage	1056	1061	1024	**1083**

(b) Comparison of operator and transition saturators.

Table 2: Per-task comparison of the initial h-value for a selection of transition saturator compositions. In each pairwise comparison, we consider the tasks for which both saturators computed 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 higher initial heuristic value than c. We use boldface to indicate the winner in the pairwise comparison (r, c) and (c, r). The bottom row holds the number of solved tasks. Saturators with * do not use $fast_t$ in the composition.

for saturated operator cost partitioning, the strongest operator saturator $perim_o{+}all_o$ from Seipp and Helmert (2019) for subset-saturated operator cost partitioning, $all_t^{*,-\infty}$ for saturated transition cost partitioning (Keller et al. 2016), and the strongest transition saturator $perim_t{+}all_t$ from Table 2a for subset-saturated transition cost partitioning.

Confirming earlier results, using subset-saturation and allowing general transition cost functions usually leads to more accurate heuristics compared to the plain all_o operator saturator. We now see that the combination of both generalizations increases the accuracy of the resulting heuristics even further. While the subset-saturation generalization only leads to a small increase in overall coverage and the transition generalization even reduces overall coverage compared to all_o, subset-saturated transition cost partitioning solves more tasks in total than all other variants. This shows that indeed allowing for more economical transition cost functions leads to computing stronger cost-partitioned heuristics.

Coverage	all_o	$\text{all}_t^{*,-\infty}$	perim_o+all_o	perim_t+all_t
airport (50)	26	15	**35**	**35**
elevators (50)	39	**42**	39	**42**
floortile (40)	**6**	**6**	3	4
freecell (80)	**66**	57	**66**	65
ged (20)	15	**19**	15	**19**
hiking (20)	13	**15**	13	**15**
logistics98 (35)	**8**	7	**8**	**8**
miconic (150)	112	**114**	112	**114**
mprime (35)	**29**	**29**	28	**29**
nomystery (20)	**20**	19	**20**	19
parcprinter (50)	30	30	32	**34**
parking (40)	**16**	14	**16**	15
pipesworld-tankage (50)	16	16	**17**	**17**
scanalyzer (50)	**23**	17	**23**	**23**
snake (20)	**14**	**14**	13	13
spider (20)	**16**	6	**16**	**16**
termes (20)	12	**13**	12	**13**
tetris (17)	**11**	**11**	10	**11**
transport (70)	**32**	31	**32**	31
woodworking (50)	36	36	36	**44**
zenotravel (20)	**13**	10	12	**13**
others (920)	**503**	**503**	**503**	**503**
Sum (1827)	1056	1024	1061	**1083**

Table 3: Number of solved tasks for a selection of saturators. The $all_t^{*,-\infty}$ saturator does not use $fast_t$ in the composition and allows costs of $-\infty$.

Per-Domain Coverage Table 3 shows the coverage per domain for the set of saturators of the previous section. For example, the transition saturator $perim_t{+}all_t$ solves more tasks than its operator saturator counterpart $perim_o{+}all_o$ in 11 domains, while the opposite holds only in 4 domains. There is a tendency that transition saturators perform better in domains where optimal plans contain the same operator multiple times. This is intuitively the case because an optimal plan that contains an operator o multiple times applies o in different states, and transition saturators can assign different costs to these operator applications.

Size Limits for Transition Saturators

The representation size of the remaining transition cost function ($remain_i$ in Definition 9) grows worst-case exponential in the number of heuristics for which we apply a transition saturator. Additionally, the time needed to apply a transition saturator to an abstraction heuristic typically increases with an increasing number of abstract transitions. To control worst-case performance, we define a classifier $\mathcal{G}_k$ with parameter $k \in \mathbb{N}$. For a given abstraction heuristic h, the classifier $\mathcal{G}_k$ decides which saturator is applied to h. The classifier $\mathcal{G}_k$ maps h to the transition saturator $perim_t{+}all_t$ if the number of abstract transitions in h is smaller than k.

k	0	10^3	10^5	10^7	10^9	∞
0	–	9	51	60	55	55
10^3	**112**	–	50	61	59	59
10^5	**285**	**197**	–	20	20	20
10^7	**375**	**290**	**122**	–	6	6
10^9	**399**	**316**	**150**	**46**	–	0
∞	**400**	**317**	**151**	**47**	**1**	–
Coverage	1061	1071	1078	**1085**	**1085**	1083

Table 4: Per-task comparison of the initial h-value for a single order and classifier $\mathcal{G}_k$ for transition saturator $perim_t{+}all_t$ and operator saturator $perim_o{+}all_o$. The parameter k of $\mathcal{G}_k$ runs on the horizontal and vertical axis. In each pairwise comparison, we consider the tasks solved by both saturators. Boldface is used to indicate the winner in the pairwise comparison (r, c) and (c, r).

Otherwise, the classifier $\mathcal{G}_k$ maps h to the operator saturator $perim_o{+}all_o$.

Table 4 shows that the algorithm yields increasingly more informed heuristics with increasing parameter k because the number of heuristics on which the transition saturator is applied typically increases with k. In other words, the algorithm yields more informed heuristics with increasing parameter k because the number of heuristics for which the algorithm uses more economical saturated transition cost functions increases with k. In contrast, the number of solved tasks peaks at 1085 for $k = 10^7$ and $k = 10^9$ abstract transitions. It is interesting to see that already such a simple classifier allows us to balance heuristic accuracy and computational effort. Since the design space for such classifiers is huge, we leave it to future work to investigate this space more thoroughly.

Multiple Orders of the Heuristics

Seipp, Keller, and Helmert (2017) showed that we solve many more tasks if we compute multiple saturated cost partitioning heuristics that consider the heuristics in different orders and maximize over them during the search. We use their *diversification* algorithm to obtain a complementary set of subset-saturated transition cost partitioning heuristics.

The algorithm starts with an empty family $\mathcal{F}$ of cost-partitioned heuristics and a set $\hat{S}$ of 1000 sample states obtained with random walks (Haslum et al. 2007). Then, until we reach a diversification time limit of T seconds, we iteratively sample a new state s with a random walk and compute the cost-partitioned heuristic h for s. If h has a strictly larger heuristic estimate than any other heuristic in $\mathcal{F}$ for any state $\hat{s} \in \hat{S}$ we add h to $\mathcal{F}$. We always include the cost-partitioned heuristic tailored to the initial state and use the same classifier $\mathcal{G}_k$ from the previous experiment with the same pair of saturators to choose from.

Table 5 shows that after one second of diversification, the operator cost partitioning method $perim_o{+}all_o$ ($k = 0$) solves nearly as many tasks as the best single-order transition cost partitioning (1082 tasks). When we diversify transition cost partitionings for one second, we solve slightly more tasks (1088–1091) for $10^5 \leq k \leq 10^7$. The highest coverage of 1170 tasks is achieved with $k = 10^3$ and 100 seconds of diversification. Coverage decreases again for large k because the algorithm either fails to compute cost-partitioned heuristics for sufficiently many orders or runs out of time for the single order. We conclude that applying transition saturators for heuristics with few abstract transitions can be beneficial also when computing multiple cost-partitioned heuristics.

	T=1s	T=10s	T=100s	T=1000s
k=0	1082	1153	1169	1160
k=10^3	1085	1152	**1170**	**1161**
k=10^5	1088	1152	1167	1156
k=10^7	**1091**	**1156**	1166	1157
k=10^9	1069	1132	1138	1127
k=∞	1069	1131	1138	1127

Table 5: Number of solved tasks using multiple orders with diversification for T seconds and classifier $\mathcal{G}_k$ that chooses between transition saturator $perim_t{+}all_t$ and operator saturator $perim_o{+}all_o$. The parameter k of $\mathcal{G}_k$ runs on the vertical axis and the diversification time limit T runs on the horizontal axis. We highlight the best configuration for a fixed value of T.

Future Work

Our empirical evaluation shows that allowing the algorithm to apply either a transition saturator or its corresponding operator saturator on a heuristic in the sequence improves the tradeoff between accuracy and runtime. Hence, improving the reasoning technique that decides whether to apply the transition saturator or its corresponding operator saturator could improve cost-partitioned heuristics. Revisiting the representation of the remaining transition cost function is another promising direction. This can be done either through exact techniques such as variable reordering on the decision diagrams or through approximate techniques such as limiting the number of available buckets where each bucket represents a set of states that are mapped to the same cost value.

Conclusions

We introduced subset-saturated transition cost partitioning that unifies two generalizations of saturated operator cost partitioning: allowing to assign different costs to transitions induced by the same operator and preserving the heuristic estimates of only a subset of states in each saturated cost partitioning iteration. Our empirical evaluation shows that subset-saturated transition cost partitioning usually computes stronger cost-partitioned heuristics for a single order of the heuristics than the two previous generalizations by themselves. The increased heuristic accuracy directly translates to solving more tasks than the earlier generalizations for single saturated cost partitioning orders. When using multiple orders, already a very simple classifier for deciding between transition and operator cost partitioning suffices to obtain a strong planner.

Acknowledgments

We thank Robert Mattmüller for discussions and comments in earlier stages of this work. Dominik Drexler was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. Jendrik Seipp received funding for this work from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 817639). Moreover, he was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215. David Speck was supported by the German Research Foundation (DFG) as part of the project EPSDAC (MA 7790/1-1).

References

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

Ball, T.; Podelski, A.; and Rajamani, S. K. 2001. Boolean and Cartesian Abstraction for Model Checking C Programs. In *Proc. TACAS 2001*, 268–283.

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

Bryant, R. E. 1985. Symbolic Manipulation of Boolean Functions Using a Graphical Representation. In *Proc. DAC 1985*, 688–694.

Dijkstra, E. W. 1959. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik* 1: 269–271.

Drexler, D.; Seipp, J.; and Speck, D. 2021. Code and data for the ICAPS 2021 paper "Subset-Saturated Transition Cost Partitioning". https://doi.org/10.5281/zenodo.4588960.

Ford, L. R. 1956. *Network Flow Theory.* Santa Monica, CA: RAND Corporation.

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.; Bonet, B.; and Geffner, H. 2005. New Admissible Heuristics for Domain-Independent Planning. In *Proc. AAAI 2005*, 1163–1168.

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. 2006. The Fast Downward Planning System. *JAIR* 26: 191–246.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In *Proc. ICAPS 2007*, 176–183.

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

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.

Kissmann, P.; Edelkamp, S.; and Hoffmann, J. 2014. Gamer and Dynamic-Gamer – Symbolic Search at IPC 2014. In *IPC-8 planner abstracts*, 77–84.

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. 2013. 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.; Keller, T.; and Helmert, M. 2017. Narrowing the Gap Between Saturated and Optimal Cost Partitioning for Classical Planning. In *Proc. AAAI 2017*, 3651–3657.

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

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

Somenzi, F. 2015. CUDD: CU decision diagram package - release 3.0.0. https://github.com/ivmai/cudd. Accessed: 2020-02-20.

Speck, D.; Geißer, F.; and Mattmüller, R. 2018. 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*, 105–109.

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