ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230377

Sensitivity Analysis for Saturated Post-Hoc Optimization in Classical Planning

Paul Höft, David Speck and Jendrik Seipp
{paul.hoft, david.speck, jendrik.seipp}@liu.se

Linköping University, Sweden

Abstract. Cost partitioning is the foundation of today's strongest heuristics for optimal classical planning. However, computing a cost partitioning for each evaluated state is prohibitively expensive in practice. Thus, existing approaches make an approximation and compute a cost partitioning only for a set of sampled states, and then reuse the resulting heuristics for all other states evaluated during the search. In this paper, we present *exact* methods for cost partitioning heuristics based on linear programming that fully preserve heuristic accuracy while minimizing computational cost. Specifically, we focus on saturated post-hoc optimization and establish several sufficient conditions for when reusing a cost partitioning computed for one state preserves the estimates for other states, mainly based on a sensitivity analysis of the underlying linear program. Our experiments demonstrate that our theoretical results transfer into practice, and that our exact cost partitioning algorithms are competitive with the strongest approximations currently available, while usually requiring fewer linear program evaluations.

1 Introduction

The strongest admissible heuristics for state space search are based on *cost partitioning* [16, 30], a technique that divides the action costs between several component heuristics, allowing for the admissible addition of their estimates. The strongest heuristic estimates are obtained by computing a cost partitioning for each evaluated state. For *landmark* heuristics, such a strategy is feasible for both optimal and suboptimal cost partitionings [13]. However, for *abstraction* heuristics, even suboptimal cost partitioning algorithms executed for each state evaluation can be computationally prohibitive [25].

An extreme example of this computational intensity are cost partitioning heuristics based on linear programming, such as *optimal cost partitioning* [15, 18], the state equation heuristic [2], and *(saturated) post-hoc optimization* [20, 26]. While it is slow but feasible to solve a linear program (LP) for the state equation or (saturated) post-hoc optimization for each state, even computing a single optimal cost partitioning can be out of reach for large sets of fine-grained abstractions [24].

As a result, previous research has leveraged state sampling either before [14, 27, 25, 5, 19] or during [22] the search. These sampled states serve as a basis for computing cost partitioning heuristics, which are then maximized over during the search process. While this strategy decreases search times for common benchmark tasks, it comes at the expense of heuristic accuracy. Moreover, it hinges on the sampled states being representative of the search space.

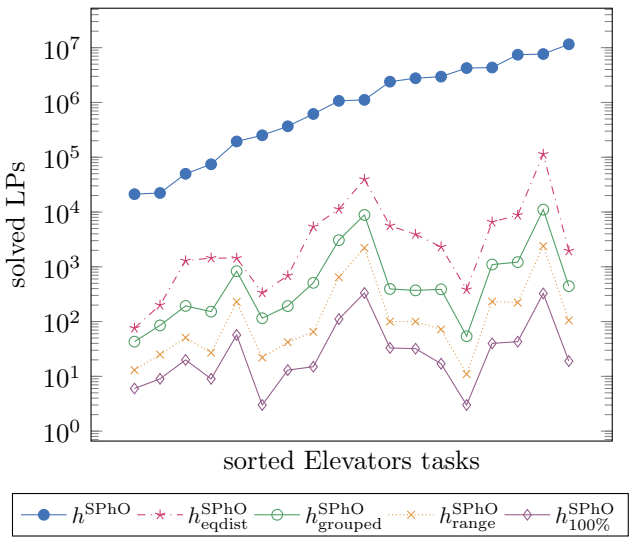


Figure 1: Number of solved LPs in the Elevators domain from the optimal track of IPC 2008 on a log scale. h^{SPhO} solves an LP for each state, while the four new variants avoid redundant computations.

Instead of relying on approximations, our study focuses on developing *exact* methods for computing a cost partitioning heuristic while minimizing computational effort. The motivation behind this is twofold: to fully preserve heuristic accuracy and to eliminate the dependency on the representativeness of sampled states within the search space.

For our analysis, we consider saturated post-hoc optimization (SPhO), an LP-based cost partitioning algorithm first introduced by Pommerening et al. [20] and later enhanced by Seipp et al. [26], who tightened the LP constraints by calculating saturated cost functions for each component heuristic. Intuitively, SPhO computes a weight for each component heuristic and the overall heuristic is the weighted sum of component heuristic values.

The basis for our work is our finding that SPhO computes very few distinct cost partitionings for the states evaluated during search in many of the standard benchmark planning problems. On average, only 0.04% of evaluated states need a new cost partitioning. Although this phenomenon has been observed for other types of cost partitioning methods such as the state equation heuristic [27], so far it has not been exploited to obtain faster algorithms. This prompts the question of whether it is possible to efficiently predict whether a new state requires a new cost partitioning or whether we can reuse

a previously computed one without sacrificing heuristic accuracy. If such a prediction is possible, then we can avoid many redundant LP computations. We show that indeed such predictions can be made by examining the SPhO LP and leveraging the well-established area of *sensitivity analysis* for linear programs [29].

We introduce a progression of four methods for deciding when a cost partitioning can be reused. Each subsequent method is capable of identifying a larger number of unnecessary computations than its predecessor. Figure 1 compares these four algorithms against solving an LP in every state (h^{SPhO}) and shows that the theoretical dominance directly translates into practice. Across all benchmarks, our experimental results reveal that up to 99.99% of cost partitioning calculations can be skipped in some domains, which makes heuristic evaluations faster by up to two orders of magnitude.

Finally, we present an offline version of SPhO that pre-samples states for which to compute cost partitionings. While our exact methods already solve more tasks than the state of the art for SPhO, this offline variant further raises the total coverage score.

2 Background

In this section, we formally introduce linear programming and sensitivity analysis and define the basics of classical planning and the saturated post-hoc optimization heuristic.

2.1 Linear Programming

Our study focuses on the saturated post-hoc optimization heuristic, which is based on linear programming. Linear programming [29] is a field of operations research that studies the definition and solution of systems of linear inequalities called *linear programs*.

Definition 1. *A linear program (LP) in* canonical *form is defined as:*

$$\begin{aligned} &\textit{Maximize } z = c^T x \textit{ subject to} \\ &Ax \leq b \\ &x \geq 0 \\ &\textit{with } A : m \times n, |b| = m, |c| = n, |x| = n \end{aligned} \tag{1}$$

In a canonical LP, the *decision variables* x optimize the *objective function* z subject to the rank m of the *coefficient matrix* A, the *right-hand side* (RHS) constraints b, and the *objective coefficients* c. Every (primal) LP has a dual formulation, which can be formed by flipping the optimization criterion and rearranging the LP to the form $v = b^T y$ subject to $A^T y \geq c, y \geq 0$ where y are the new decision variables. In the following, we assume that all LPs or their duals are in canonical form.

Linear programming is a versatile tool due to its broad applicability and polynomial complexity [28, 17]. To solve an LP it is converted into *standard* form by adding m *slack* variables to the decision variables so that Equation (1) becomes an equality. The most commonly used algorithm for solving LPs is the *simplex* algorithm [4]. The space of *feasible* solutions, i.e., the assignments to x that satisfy all constraints, forms a convex polytope. The simplex algorithm can be understood as traversing the extreme points of this polytope. Each of these extreme points, and thus each solution given by the simplex algorithm, divides the decision variables into two categories: m *basic* and n *non-basic* variables when also counting slack variables. The basic variables form the linear basis of the current solution and are the variables that can take non-zero values for this solution. Each basic variable is associated with one of the constraints, and we say that x_i is basic for that constraint. All other variables are called non-basic and have a value of zero.

2.2 Sensitivity Analysis of Linear Programs

Sensitivity analysis, also known as *post-optimality analysis*, determines how sensitive the optimal solution of an LP is to small changes in its parameters [7]. For each non-basic variable, the simplex algorithm provides additional sensitivity information that can be used to determine the effect of small changes in that variable on the current basis. We distinguish between non-basic slack and non-basic non-slack variables. For non-basic non-slack variables x_i, this information is the *opportunity cost* $\bar{c}_i$.

Definition 2. *The opportunity cost $\bar{c}_i$ of a non-basic non-slack variable x_i is the increase for c_i at which x_i would enter the optimal basis. It is also the reduction of z that would occur when setting $x_i = 1$, provided this is feasible.*

For non-basic slack variables, the simplex algorithm produces similar information in the form of the *shadow prices* $\bar{y}_i$.

Definition 3. *The shadow price $\bar{y}_i$ of a non-basic slack variable x_i is the change in z that occurs when changing the RHS constraint for this slack variable b_{i-n} by ± 1, provided this is feasible.*

Intuitively, opportunity costs are the influence of a small increase of the non-basic variables on the objective value and shadow prices are the influence of small variations of the RHS constraints on the objective value. Note that opportunity costs are one-sided while shadow prices are valid in both directions. The simplex algorithm also transforms the RHS constraints into new values, which we call $\bar{b}$.

For our methods below, we extract intervals $[L, U]$, with $L, U \in \mathbb{R} \cup \{-\infty, \infty\}$, from an optimal LP solution such that if objective coefficients c_j or constraint values b_j remain within this interval, the basis of the LP (i.e., which variables are basic) remains unchanged. Even if we maintain the optimal basis and change an objective coefficient or constraint value only within such an interval, the optimal objective value may change. However, having all this information, there are cheap approaches to recompute the new optimal objective value of the LP without solving it again. This will be sufficient to check whether a new cost partitioning is required for a given state.

Next, we briefly describe two ways to compute such interval ranges that preserve the optimal basis and allow for cheap recomputation of the new optimal objective value of the LP. The definitions presented are, with slight modifications, also valid for LPs in dual canonical form. The definitions below assume $\frac{n}{\infty} = \frac{n}{-\infty} = 0$ for all finite n.

2.2.1 Ranges of Right-Hand Side Constraints

The allowed ranges for RHS values of a primal LP solution can be extracted from the modified RHS values $\bar{b}$ as follows:

Theorem 1. *[4] An LP basis remains optimal when changing one RHS bound from b_k to b'_k, as long as $L \leq b'_k \leq U$, where*

$$L = b_k - \max_r \left\{ \frac{-\bar{b}_r}{a_{rk}} \middle| a_{rk} > 0 \right\}, U = b_k + \min_r \left\{ \frac{-\bar{b}_r}{a_{rk}} \middle| a_{rk} < 0 \right\}.$$

The new *optimal* objective value z' can be computed from the shadow prices of the RHS constraints: $z' = z + \bar{y}_i(b'_k - b_k)$, where $\bar{y}_i$ is the shadow price of the slack variable for constraint k.

2.2.2 100% Rule

The sensitivity intervals $[L, U]$ are limits derived for single perturbations of coefficients or bounds. This means that, without further consideration, they only hold as long as only one such parameter is changed at a time. As soon as multiple parameters are changed at the same time, these bounds are no longer valid. To address this issue, the 100% rule has been proposed, which provides a set of safe conditions for generalizing the interval ranges to cases where multiple parameters are changed simultaneously.

Theorem 2. *[4] An LP basis remains optimal while changing multiple RHS constraints from b_k to $b'_k = b_k + \Delta b_k$ as long as*

$$\sum_{0 \leq k \leq n} \lambda_k \leq 1 \text{ and } \lambda_k = \frac{\Delta b_k}{\Delta b_k^*} \text{ with } \Delta b_k^* = \begin{cases} U - b_k & \text{if } \Delta b_k \geq 0 \\ L - b_k & \text{otherwise.} \end{cases}$$

The change in objective value can be calculated as the sum of change given by the shadow prices: $\Delta z = \sum_{i=1}^{n} \bar{y}_i \Delta b_{i-n}$, so $z' = z + \Delta z$.

There are methods to compute stronger approximations in the case of multiple parameter changes, such as *parametric analysis* [4] and *two-sided* shadow prices [7]. Since these methods are quite sophisticated and computationally demanding, we focus on the two efficient sensitivity analysis methods presented above.

2.3 Classical Planning

We consider planning tasks in the SAS$^+$ formalism [1], where a task consists of a set of finite domain variables that induce a set of states, a finite set of operators (or actions), an initial state, a goal condition, and a non-negative function that describes the cost of applying each operator. The details of planning tasks are not relevant for the technical contribution of this paper; all that matters is that each planning task compactly encodes a weighted transition system.

Definition 4. *A weighted* transition system $\mathcal{T} = \langle S, L, T, cost, s_0, S_* \rangle$ *is a directed labeled graph defined by a finite set of* states S*, a finite set of* labels L*, a finite set of* labeled transitions T*: $s \xrightarrow{\ell} s'$ with $s, s' \in S$ and $\ell \in L$, a* cost-function $cost : L \to \mathbb{R} \cup \{-\infty, \infty\}$ *that assigns a cost to each label, an* initial state $s_0 \in S$*, and a set of* goal states $S_* \subseteq S$*. We also write $s \in \mathcal{T}$ for $s \in S$.*

Note that we consider real-valued costs for transition systems, although planning tasks usually have non-negative costs, to allow for the later definition of minimum saturated cost functions. A *goal path* for a state s, called a *plan*, is a sequence of transitions leading from s to any goal state. We focus on optimal planning, where the objective is to find a cost-optimal plan π for the initial state of a given planning task. Heuristic search [3] is one of the main ways of solving planning tasks optimally. This usually involves running an A* search [8] with an admissible heuristic on the transition system. A *heuristic* is a function h: $S \to \mathbb{R} \cup \{-\infty, \infty\}$ that estimates the cost of the cheapest goal path for a given state. The perfect heuristic $h^*_{\mathcal{T}}$ maps each state s in $\mathcal{T}$ to the cost of the cheapest goal path from s or to ∞ if there is no goal path from s. A heuristic h is *admissible* if it never overestimates the true cost of the shortest goal path: $h(s) \leq h^*(s)$ for all states $s \in S$.

One of the strongest families of admissible heuristics are *abstraction heuristics* [6, 11, 16, 23]. The idea behind abstraction heuristics is to simplify the transition system $\langle S, L, T, cost, s_0, S_* \rangle$ of a planning task with a surjective *abstraction function* $\alpha : S \to S^\alpha$, yielding an *abstract transition system* $\mathcal{T}^\alpha : \langle S^\alpha, L, T^\alpha, cost, \alpha(s_0), S^\alpha_* \rangle$, where $T^\alpha = \{\alpha(s) \xrightarrow{\ell} \alpha(s') \mid s \xrightarrow{\ell} s' \in T\}$ and $S^\alpha_* = \{\alpha(s) \mid s \in S_*\}$. For a concrete state s, the abstraction heuristic h^α yields the perfect goal distance of the abstract state $\alpha(s)$ in the abstract transition system $\mathcal{T}^\alpha$. The size of $\mathcal{T}^\alpha$ is chosen to be small enough for all abstract goal distances to be efficiently computable.

Cost partitioning [15, 16] allows for *adding* multiple abstraction heuristics admissibly, which can be significantly more accurate than *maximizing* over them. A cost partitioning for a transition system $\mathcal{T}$ and a tuple of abstraction heuristics $H = \langle h^{\alpha_1}, \ldots, h^{\alpha_n} \rangle$ is a tuple of cost functions $\langle cost_1, \ldots, cost_n \rangle$ such that $\sum_{i=1}^{n} cost_i(\ell) \leq cost(\ell)$ for all $\ell \in L$. It defines the cost partitioning heuristic $h(s) = \sum_{i=1}^{n} h^{\alpha_i}(s, cost_i)$, where $h^{\alpha_i}(s, cost_i)$ is the abstraction heuristic h^{α_i} evaluated under cost function $cost_i$.

2.4 Saturated Post-hoc Optimization

The post-hoc optimization heuristic is an admissible heuristic for classical planning from the family of operator counting heuristics [21]. For a given state s, post-hoc optimization over abstraction heuristics H minimizes the number of times each operator is counted while ensuring that no heuristic value $h(s)$ decreases for any abstraction heuristic $h \in H$.

In this paper, we consider the *saturated* post-hoc optimization (SPhO) heuristic, because it is stronger than its non-saturated counterpart in theory and practice [26]. A crucial part of SPhO are *minimum saturated cost functions*:

Definition 5. *[25] The minimum saturated cost function mscf of an abstract transition system $\mathcal{T}'$ with transitions T is defined as*

$$mscf(\ell) = \sup_{a \xrightarrow{\ell} b \in T} (h^*_{\mathcal{T}'}(a) \ominus h^*_{\mathcal{T}'}(b)),$$

where the $\ominus$ operator is defined as regular subtraction for finite values, and for infinite values we use $x \ominus y = -\infty$ iff $x = -\infty$ or $y = \infty$ and $x \ominus y = \infty$ iff $x = \infty \neq y$ or $x \neq -\infty = y$.

Definition 6. *Given a transition system $\mathcal{T} = \langle S, L, T, cost, s_0, S_* \rangle$ and a tuple of abstraction heuristics H for $\mathcal{T}$, the heuristic value $h^{SPhO}(s)$ for a state s is the* objective value *of the* SPhO LP*:*

$$\begin{aligned} &\text{minimize} \sum_{\ell \in L} cost(\ell) \cdot Y_\ell \text{ s.t.} \\ &\sum_{\ell \in L} mscf_h(\ell) \cdot Y_\ell \geq h(s) \text{ for all } h \in H \\ &Y_\ell \geq 0 \text{ for all } \ell \in L \end{aligned} \tag{2}$$

By h^{SPhO} we denote the heuristic that solves the SPhO LP for each evaluated state. The SPhO LP implicitly computes a cost partitioning through its dual LP [20]. The dual objective coefficients are the shadow prices $\bar{y}_i$, which therefore give the heuristic scaling factors of the cost partitioning heuristic: $h^{\text{SPhO}}(s) = \sum_{i=1}^{n} \bar{y}_i h_i(s)$.

3 Reusing Solved LPs for SPhO

To avoid solving the SPhO LP from Definition 6 for each evaluated state, we formulate four *cover rules* for SPhO. These rules decide whether an LP solution can be reused for a different state. Given two states s, s' and an SPhO LP solution *sol* for s, we say that *sol covers*

Algorithm 1 Lazy saturated post-hoc optimization. For a given state s, check whether any of the previously computed solutions to the SPhO LP covers s. If so, (possibly) adapt the solution to s and return the adapted objective value. Otherwise, solve the SPhO LP for s and store the solution.

```
Sols ← ∅
function LazySPhO(s)
    if Covers(sol, s) for any sol ∈ Sols then
        return Adapt(sol, s)
    ⟨sol, h⟩ ← solve SPhO LP for s
    Sols ← Sols ∪ {sol}
    return h
```

s' if its basis is provably optimal for s'. In this case, we can skip solving the SPhO LP for s' and instead compute $h^{\text{SPhO}}(s')$ from *sol*.

To use the cover rules during an A* search, we introduce a variant of the SPhO heuristic, called *lazy* SPhO, which is parameterized by a cover rule and only solves LPs for states that are not covered by any previously obtained LP solutions. Algorithm 1 shows pseudo-code.

We now present the four cover rules that can be used to implement the Covers function in Algorithm 1. Each subsequent rule is stronger than the previous one, allowing us to reuse a solution for a larger set of states. The first two cover rules for SPhO are based on a structural analysis of the SPhO LP.

3.1 Equal Abstract Goal Distances: h^{SPhO}_{eqdist}

One simple way to detect that an LP solution *sol* for s covers a different state s', is by checking whether the two states induce the same SPhO LP (Definition 6). When optimizing the SPhO LP for s and s', the only LP parameters that change are the abstract goal distances. Thus, if the two states have the same abstract goal distances, the resulting LPs will be identical.

Cover Rule 1. *Let $\mathcal{T}$ be a transition system and H a tuple of heuristics for $\mathcal{T}$. If sol is a solution of the SPhO LP for state $s \in \mathcal{T}$, then sol covers state $s' \in \mathcal{T}$ if $h(s) = h(s')$ for all $h \in H$. The objective value of sol can be used as the heuristic value for s' without adaptation.*

By $h^{\text{SPhO}}_{\text{eqdist}}$ we denote the lazy SPhO heuristic using Cover Rule 1.

Theorem 3. $h^{SPhO}_{eqdist} = h^{SPhO}$.

Proof. $h^{\text{SPhO}}_{\text{eqdist}}(s) = h^{\text{SPhO}}(s)$ for all states s where Cover Rule 1 is inapplicable. In states s' where it is active, the skipped LP for s' is equal to a previously computed LP for a state s, so $h^{\text{SPhO}}(s') = h^{\text{SPhO}}(s)$ and since Adapt preserves the value for s', we have $h^{\text{SPhO}}_{\text{eqdist}}(s') = h^{\text{SPhO}}(s)$ and thus $h^{\text{SPhO}}_{\text{eqdist}}(s') = h^{\text{SPhO}}(s')$. □

We demonstrate Cover Rule 1 and the rules below with the example transition system $\mathcal{T}$ and three abstractions for $\mathcal{T}$ in Figure 2.

Intuitively, Cover Rule 1 avoids unnecessary LP computations when the used abstractions cannot distinguish between two states based solely on goal distances. States s_2 and s_3 in Figure 2 exemplify this: all three abstractions fail to distinguish s_2 from s_3. Consequently, $h^{\text{SPhO}}_{\text{eqdist}}$ skips computing the LP for s_3 when encountering s_3 during the A* search, because its abstract goal distances (2, 0 and 1) are the same as those for s_2, which has already been evaluated.

3.2 Grouped Constraints: $h^{SPhO}_{grouped}$

Another key component of the SPhO LPs are the minimum saturated cost functions. Besides the abstract goal distances, they are the only information used from the abstractions to build the SPhO LPs. Therefore, if two abstraction heuristics h_1 and h_2 have the same minimum saturated cost function, the resulting LP will contain a duplicated row in the coefficient matrix A. The LP for state s stays mathematically equal if we only consider the heuristic among h_1 and h_2 with the higher estimate for s, as this heuristic incurs a tighter bound than the other. This is equivalent to combining both constraints and using $\max(h_1(s), h_2(s))$ as the RHS bound. Thus, by partitioning all abstractions into sets H_j of heuristics with equal minimum saturated cost functions, Equation 2 can be simplified to generate fewer constraints. Also, we can exclude the partition H_0, containing all abstractions with a minimum saturated cost function that always returns 0, since they add no information to the LP anyway.

Definition 7. *The* grouped *SPhO LP is:*

$$\begin{aligned} &\textit{minimize} \sum_{\ell \in L} cost(\ell) \cdot Y_\ell \textit{ s.t.} \\ &\sum_{\ell \in L} mscf_h(\ell) \cdot Y_\ell \geq \max_{h \in H_j} h(s) \textit{ for } 1 \leq j \leq m \qquad (3) \\ &Y_\ell \geq 0 \textit{ for all } \ell \in L \end{aligned}$$

Solutions to the *grouped* SPhO LP always have the same objective value as solutions to the vanilla SPhO LP because only redundant constraints are left out. A constraint of the form $lhs \geq a$ is always more restrictive than a constraint $lhs \geq b$ if $a > b$ when minimizing, as the possible values for *lhs* in the second constraint are a subset of the possible values of the first constraint. Using the abstraction grouping we can define a new cover rule.

Cover Rule 2. *Let $\mathcal{T}$ be a transition system and H a tuple of heuristics for $\mathcal{T}$. Furthermore, let $\bigcup_{j=1}^{m} H_j = H$ be a partitioning of all abstractions H into disjoint subsets H_j, such that all heuristics h in each partition H_j have the same minimum saturated cost function. Finally, let sol be a solution of the SPhO LP for state $s \in \mathcal{T}$. Then sol covers state $s' \in \mathcal{T}$ if $\max_{h \in H_j} h(s) = \max_{h \in H_j} h(s')$ for all partitions H_j. The objective value of sol can be used as the heuristic value for s' without adaptation.*

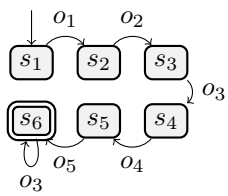
By $h^{\text{SPhO}}_{\text{grouped}}$ we denote the lazy SPhO heuristic using Cover Rule 2.

Theorem 4. $h^{SPhO}_{grouped} = h^{SPhO}$.

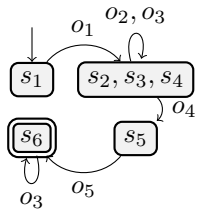
Proof. For all states s, the LP for $h^{\text{SPhO}}_{\text{grouped}}(s)$ is equivalent to the LP for $h^{\text{SPhO}}_{\text{eqdist}}(s)$ if we group constraints for the latter. The claim follows from Theorem 3 and the fact that grouping constraints, as outlined above, does not change the objective value of an LP. □

Grouping abstractions by their minimum saturated cost function has several benefits. First, it allows for detecting more unnecessary LP solver calls than the previous cover rule. Second, it reduces the size of the LP, leading to faster solving times. Finally, it can lead to better sensitivity ranges, as shown in the next section. Thus, we also base the two remaining cover rules on the grouped SPhO LP.

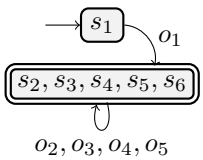
In the running example from Figure 2, Cover Rule 2 needs to solve one LP less, because abstractions A and C have the same minimum saturated cost function. We can skip solving an LP for s_4 because states s_2 and s_4 have the same maximum abstract goal distance in the two abstraction partitions $\{A, C\}$ ($\max(2, 1) = \max(2, 2)$) and $\{B\}$ (0).



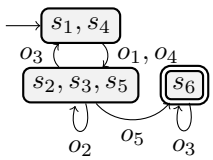
(a) Concrete transition system



(b) Abstraction A



(c) Abstraction B



(d) Abstraction C

	h^A	h^B	h^C	$\max(h^A, h^C)$
s_1	3	1	2	3
s_2	2	0	1	2
s_3	2	0	1	2
s_4	2	0	2	2
s_5	1	0	1	1

(e) Heuristic values

	cost	$mscf_A$	$mscf_B$	$mscf_C$
o_1	1	1	1	1
o_2	1	0	0	0
o_3	1	0	0	0
o_4	1	1	0	1
o_5	1	1	0	1

(f) Cost functions

			RHS		
state	abstraction	shadow price	L	value	U
s_1	h^A	1	2	3	∞
	h^B	0	$-\infty$	1	3
	h^C	0	$-\infty$	2	3
s_1	$\max(h^A, h^C)$	1	1	3	∞
	h^B	0	$-\infty$	1	3
s_2	$\max(h^A, h^C)$	1	0	2	∞
	h^B	0	$-\infty$	0	0

(g) LP solutions for selected states, plus sensitivity information.

Figure 2: Example task and abstractions for showcasing cover rules.

3.3 Sensitivity Analysis using RHS Constraints: h^{SPhO}_{range}

The two cover rules presented above are limited to states with the same SPhO heuristic value. However, sensitivity analysis (SA) (Section 2.2) allows us to check whether a basis of an LP solution computed for state s remains optimal for a new state s', which we can directly transform into cover rules that can detect unnecessary LP computations even if s and s' have different SPhO heuristic values.

Using the sensitivity analysis formula from Definition 1 we obtain another cover rule:

Cover Rule 3. *Let $\mathcal{T}$ be a transition system and H a tuple of heuristics for $\mathcal{T}$. An LP solution sol for state s covers s' if for at least $|H| - 1$ abstractions h we have $h(s) = h(s')$ and for at most one abstraction h' with $h'(s) \neq h'(s')$ we have $L \leq h'(s') \leq U$, where L and U are the lower and upper bounds of the sensitivity range for h'. Then, the new objective value of sol for s' is $z + \bar{y}_i(h'(s') - h'(s))$, where $\bar{y}_i$ is the shadow price belonging to the slack variable for the constraint of h'.*

By $h^{\text{SPhO}}_{\text{range}}$ we denote the lazy SPhO heuristic on the grouped SPhO LP using Cover Rule 3.

Theorem 5. $h^{SPhO}_{range} = h^{SPhO}$.

Proof. $h^{\text{SPhO}}_{\text{range}}(s)$ equals h^{SPhO} for all states s where Cover Rule 3 is inactive by construction. In states s' where it is active the sensitivity analysis rule for RHS values (Theorem 1) guarantees that the basis of $sol = h^{\text{SPhO}}_{\text{range}}(s)$ is the same as the basis of $sol' = h^{\text{SPhO}}_{\text{range}}(s')$ making the adapted solution $h^{\text{SPhO}}_{\text{range}}(s') = h^{\text{SPhO}}(s)$. □

Cover Rule 3 is a generalization of Rule 2: Whereas Rule 2 requires *all* heuristic estimates to be the same for two states, Rule 3 allows for a bounded change of the estimate made by one heuristic. The running example in Figure 2 shows a situation where Cover Rule 3 leads to fewer solved LPs by using the information from sensitivity analysis. The rule detects that the LP solution *sol* for s_2 covers s_5 because $h^B(s_2) = h^B(s_5)$ and while the value $\max(h^A, h^C)$ for heuristic partition $\{A, C\}$ changes from 2 to 1 between s_2 and s_5, it stays inside the sensitivity analysis interval $[0, +\infty)$ (see bottom part of Table 2g).

3.4 Sensitivity Analysis using 100% Rule: $h^{SPhO}_{100\%}$

For the final cover rule, we consider the 100% rule from Definition 2, which yields a generalization of Cover Rule 3.

Cover Rule 4. *Let $\mathcal{T}$ be a transition system, H a tuple of m heuristics for $\mathcal{T}$, and s, s' two states in $\mathcal{T}$. Also, let sol be a solution of the SPhO LP for s, let b_j and b'_j be the RHS values of constraint j in the SPhO LPs for s and s', and let $\Delta b_j = b'_j - b_j$. Then sol covers s' if*

$$\sum_{0 \leq j \leq m} \lambda_j \leq 1 \text{ where } \lambda_j = \frac{\Delta b_j}{\Delta b^*_j}, \Delta b^*_j = \begin{cases} U_j - b_j & \text{if } \Delta b_j \geq 0 \\ L_j - b_j & \text{otherwise.} \end{cases}$$

The objective value of sol has to be adapted for state s' to $z' = z + \sum_{j=0}^{m} \bar{y}_{j+n} \Delta b_j$.

By $h^{\text{SPhO}}_{100\%}$ we denote the lazy SPhO heuristic on the grouped SPhO LP using Cover Rule 4.

Theorem 6. $h^{SPhO}_{100\%} = h^{SPhO}$.

Proof. $h^{\text{SPhO}}_{100\%}(s) = h^{\text{SPhO}}(s)$ for all states s where Cover Rule 4 is inapplicable. When the rule decides that a state s' is covered by the LP solution *sol* for state s, the 100% rule from Theorem 2 guarantees that the basis of *sol* is the same as the basis of the corresponding LP solution for s'. By adapting the objective value as per Theorem 2 we obtain $h^{\text{SPhO}}_{100\%}(s') = h^{\text{SPhO}}(s')$. □

Since this rule also detects when a *single* RHS value changes within its sensitivity range, Rule 4 is a generalization of Rule 3.

As an example, we again consider the task from Figure 2 and the sensitivity information for state s_1. Here, Rule 4 detects that the LP solution for s_1 is reusable for all other states, making all but the first LP solution redundant. This is the case because both considered heuristic values ($\max(h^A, h^C)$ and h^B) stay inside their sensitivity intervals $[1, +\infty)$ and $(-\infty, 3]$ for all states s_2 to s_5. Note that the lower bound for changes in h_B is $-\infty$, which means that decreases for this value do not contribute to the 100% limit, as we defined $\frac{n}{-\infty} = 0$ for finite n.

The example additionally illustrates the utility of grouping heuristics with the same minimum saturated cost function, as the lower

bound for abstraction A is worse without grouping (see Table 2g). Without grouping, we would need to solve the LP for s_5.

Note that even the most general Cover Rule is only a sufficient condition for skipping LP computations and it is unable to detect all possible cases in which a basis might be optimal for other states. The 100% range is only a safe approximation of the allowed bounds when multiple parameters change and the real bound ranges for simultaneous changes can be larger than the ones allowed by the 100% rule.

4 Offline Saturated Post-hoc Optimization

For other cost partitioning algorithms, such as optimal cost partitioning [14], potential heuristics [27] and saturated cost partitioning [24], it is preferable to precompute cost partitionings for a set of sample states and maximize over the resulting estimates during the search, rather than computing a cost partitioning for each evaluated state. Since this approach has not been applied to (saturated) post-hoc optimization before, we introduce it here as a baseline that *approximates* h^{SPhO}. To this end, we build our new $h^{\text{SPhO}}_{\text{offline}}$ heuristic with the diversification procedure from Seipp et al. [25]: We begin by sampling $n = 1000$ states [9] and initializing an empty set $\mathcal{C}$ of cost partitionings. Then, for $t = 200$ seconds, we iteratively sample a new state s, solve the SPhO LP for s, and add the resulting cost partitioning to $\mathcal{C}$ if it yields a higher estimate for any of the n sample states than all cost partitionings that are already part of $\mathcal{C}$, and discard it otherwise.

5 Experiments

To evaluate if our theoretical results transfer into practice, we implemented the lazy SPhO heuristic with all four cover rules and the offline SPhO heuristic in the Scorpion planner [25], which is an extension of Fast Downward 22.12 [10] and contains an implementation of the eager SPhO heuristic [26]. In our experiments, we run A* searches [8] with all SPhO heuristic variants over pattern database heuristics [6] using patterns of sizes 1 and 2 that are *interesting* [20, 19] for general cost partitioning [18]. We use CPLEX 20.1 to solve the linear programs and evaluate the different approaches on the 1827 benchmark tasks without conditional effects from the optimal tracks of the 1998–2018 International Planning Competitions (IPCs). For resource limits, we adhere to the standard IPC setting with a runtime of 30 minutes and 8 GiB of memory. All benchmarks, source code, and experimental data are available online [12].

In the following, we will first analyze the number of solved LPs required by the different SPhO heuristic variants, before turning to the performance in terms of solved problems and runtime.

5.1 Number of Solved LPs

Figure 3 shows a per-problem comparison of the number of LP computations needed by the new lazy SPhO heuristic in comparison to the eager SPhO heuristic from the literature. We see that all of our approaches reduce the number of LPs to solve by several orders of magnitude, while fully preserving heuristic accuracy. Each of our presented approaches reduces the number of necessary LP computations in comparison to the previous rule. $h^{\text{SPhO}}_{\text{eqdist}}$ needs fewer LP solver calls than h^{SPhO} in all 817 tasks solved by h^{SPhO}, $h^{\text{SPhO}}_{\text{grouped}}$ needs fewer LP computations than $h^{\text{SPhO}}_{\text{eqdist}}$ for 55 tasks, $h^{\text{SPhO}}_{\text{range}}$ incurs solving fewer LPs than $h^{\text{SPhO}}_{\text{grouped}}$ for 768 tasks and $h^{\text{SPhO}}_{100\%}$ reduces the number of LP solver calls for 654 tasks compared to $h^{\text{SPhO}}_{\text{range}}$.

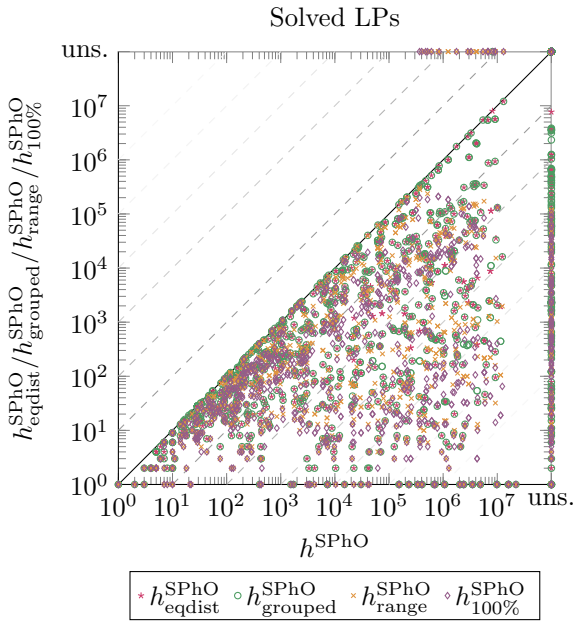


Figure 3: A comparison of the number of solved LPs between the eager SPhO heuristic h^{SPhO} and all presented lazy versions on a logarithmic scale. Tasks that are not solved within the resource limits appear on "uns." axes.

h^{SPhO}	$h^{\text{SPhO}}_{\text{eqdist}}$	$h^{\text{SPhO}}_{\text{grouped}}$	$h^{\text{SPhO}}_{\text{range}}$	$h^{\text{SPhO}}_{100\%}$	$h^{\text{SPhO}}_{\text{offline}}$
817	940	946	887	905	**969**

Table 1: Number of solved tasks (out of 1827) by the eager SPhO heuristic h^{SPhO} and our new variants.

Figure 4 shows the geometric mean number of required LP solver calls for all SPhO heuristic variants on *all* considered IPC benchmark domains and the Mystery and VisitAll domains. Note that the figure only considers tasks solved by all variants, that the number of LP computations by h^{SPhO} equals the number of evaluated states, and that $h^{\text{SPhO}}_{\text{offline}}$ always solves 1000 LPs because it uses an ensemble of 1000 states for the diversification procedure. Figure 4 shows again that our new approaches, which skip unneeded LP computations, reduce the average number of solved LPs by several orders of magnitude compared to the eager h^{SPhO} approach. However, the effectiveness of the presented cover rules, i.e., the number of LP computations we can avoid, depends on the planning domain. In the Mystery domain, we reduce the number of solved LPs by about three orders of magnitude (which is similar to the results for the Elevators domain shown in Figure 1). In the VisitAll domain, however, it is almost never possible to cheaply compute the heuristic value of a newly encountered state based on the solution of an LP solved earlier.

5.2 Coverage and Runtime

A natural question is whether the observed reduction in the number of LP solver calls to compute the SPhO heuristic is reflected in the number of solved tasks, i.e., the planner *coverage*, and the planner runtime.

Table 1 shows the total number of solved planning tasks using the different SPhO variants. Considering the online approaches, we see that all our new lazy approaches solve many more problems than eager h^{SPhO} due to the lower number of solved LPs. However, we

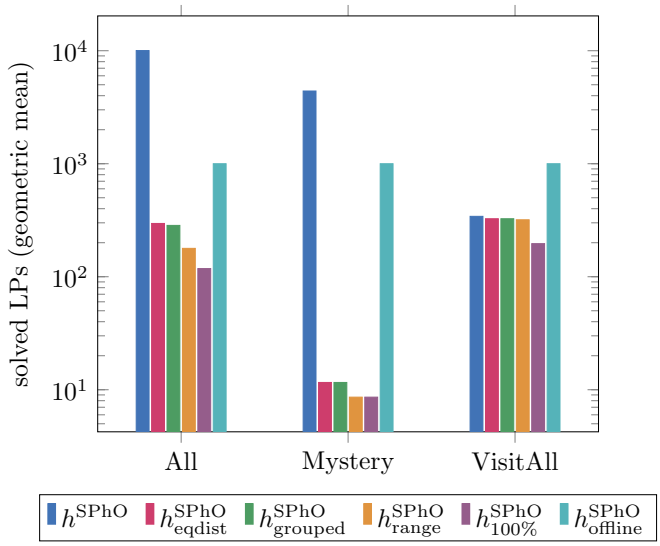


Figure 4: Comparison of the number of solved LPs (geometric mean) for eager h^{SPhO} and our lazy and offline variants on a logarithmic scale over all domains and the Mystery and VisitAll domains.

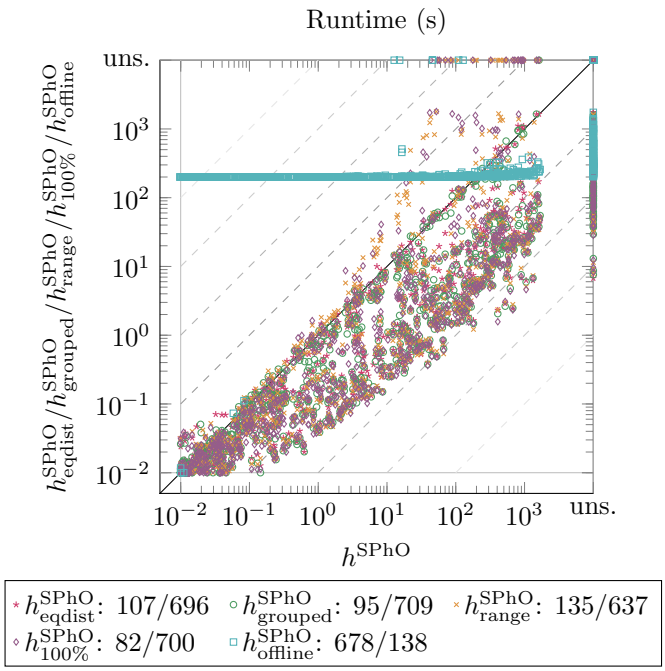


Figure 5: Runtime comparison between the eager SPhO heuristic h^{SPhO} and our lazy and offline variants on a logarithmic scale. The legend additionally shows the number of tasks a variant solves slower/faster than h^{SPhO}. Problems that are not solved within the resource limits appear on "uns." axes.

also see that the heuristic with the lowest number of LP computations, $h^{\text{SPhO}}_{100\%}$, does not solve the highest number of tasks. The reason for this is that the complexity of evaluating applicability of the different cover rules for a state varies greatly. Evaluating the $h^{\text{SPhO}}_{\text{eqdist}}$ and $h^{\text{SPhO}}_{\text{grouped}}$ heuristics requires constant overhead, regardless of how many LP solutions are stored, while the $h^{\text{SPhO}}_{\text{range}}$ and $h^{\text{SPhO}}_{100\%}$ heuristics incur a linear overhead for checking whether heuristic values fall into sensitivity intervals. This leads to $h^{\text{SPhO}}_{\text{grouped}}$ having the highest coverage score among the online variants, even though it computes more LPs than $h^{\text{SPhO}}_{\text{range}}$ and $h^{\text{SPhO}}_{100\%}$.

Interestingly, the offline variant $h^{\text{SPhO}}_{\text{offline}}$ solves the most tasks overall. While $h^{\text{SPhO}}_{\text{offline}}$ does not retain full heuristic accuracy, the ensemble of 1000 states can often cover almost all distinct cost partitionings. This, combined with the fast evaluation speed, explains its high coverage score.

Finally, we see in Figure 5 that the strong coverage performance of $h^{\text{SPhO}}_{\text{offline}}$ does not necessarily transfer to runtime, as it often takes much longer to solve a problem than standard h^{SPhO}. The reason for this is that $h^{\text{SPhO}}_{\text{offline}}$ uses a fixed precomputation time (200 seconds), after which it quickly solves the tasks that are also solved by h^{SPhO}. For the lazy approaches, the runtime is usually significantly lower than for eager h^{SPhO}.

6 Discussion

In linear programming, it is common to repeatedly solve similar LPs, and modern LP solvers such as CPLEX use a number of techniques to reuse information about the previously solved LP and its solution, sometimes referred to as *advanced* or *warm starts*. Thus, the implementation of LP-based heuristics such as eager h^{SPhO} use such warm starts and information about the previously solved LP by default, which is in line with the goals of our paper. This is also a big reason for why the huge reductions in the number of solved LPs are not reflected in speedups of the same magnitude. However, there is an important difference between the theory presented in this paper and using LP solvers with warm starts. An LP solver with warm starts stores only the last LP and information about its solution. As we have seen in the empirical evaluation, this may not be sufficient for obtaining the best performance, since we regularly encounter states that require solving or reusing information from various different previously encountered LPs that a standard LP solver has not cached.

7 Conclusions

In this work, we presented new theory on exact methods for cost partitioning heuristics based on linear programming that fully preserve heuristic accuracy while minimizing computational cost. To this end, we established a connection between the sensitivity analysis of LPs and the heuristic evaluation of newly encountered states for LP-based heuristics. In particular, we focused on the SPhO heuristic and showed that it is possible to reuse information about the solutions of LPs computed for previously encountered states, thus eliminating costly LP computations for newly encountered states while preserving full heuristic accuracy. These theoretical results translated directly into practice, where our new variants of the SPhO heuristic resulted in orders of magnitude fewer LP solver calls and a significant increase in coverage in the empirical evaluation. Finally, our new offline SPhO variant that approximates h^{SPhO} surpassed even the strongest of our exact methods in the number of solved tasks.

The application of sensitivity analysis to automated planning heuristics opens up other interesting directions for future work. It is interesting to consider parametric analysis [4] or two-sided shadow prices [7], which may lead to even stronger cover rules, and to analyze and study our theory on redundant LP computations for other LP-based heuristics such as optimal cost partitioning [15, 18] or the state equation heuristic [2].

Acknowledgements

This work was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215, and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) and the Swedish National Infrastructure for Computing (SNIC), partially funded by the Swedish Research Council through grant agreements no. 2022-06725 and no. 2018-05973.

References

[1] Christer Bäckström and Bernhard Nebel, 'Complexity results for SAS^+ planning', *Computational Intelligence*, **11**(4), 625–655, (1995).
[2] Blai Bonet, 'An admissible heuristic for SAS^+ planning obtained from the state equation', in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, ed., Francesca Rossi, pp. 2268–2274. AAAI Press, (2013).
[3] Blai Bonet and Héctor Geffner, 'Planning as heuristic search', *Artificial Intelligence*, **129**(1), 5–33, (2001).
[4] Stephen P. Bradley, Arnoldo C. Hax, and Thomas L. Magnanti, *Applied Mathematical Programming*, Addison-Wesley, 1977.
[5] Dominik Drexler, Jendrik Seipp, and David Speck, 'Subset-saturated transition cost partitioning', in *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, eds., Robert P. Goldman, Susanne Biundo, and Michael Katz, pp. 131–139. AAAI Press, (2021).
[6] Stefan Edelkamp, 'Planning with pattern databases', in *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, eds., Amedeo Cesta and Daniel Borrajo, pp. 84–90. AAAI Press, (2001).
[7] T. Gal, 'Shadow prices and sensitivity analysis in linear programming under degeneracy', *Operations-Research-Spektrum*, **8**, 59–71, (1986).
[8] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael, 'A formal basis for the heuristic determination of minimum cost paths', *IEEE Transactions on Systems Science and Cybernetics*, **4**(2), 100–107, (1968).
[9] Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig, 'Domain-independent construction of pattern database heuristics for cost-optimal planning', in *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, pp. 1007–1012. AAAI Press, (2007).
[10] Malte Helmert, 'The Fast Downward planning system', *Journal of Artificial Intelligence Research*, **26**, 191–246, (2006).
[11] Malte Helmert, Patrik Haslum, and Jörg Hoffmann, 'Flexible abstraction heuristics for optimal sequential planning', in *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, eds., Mark Boddy, Maria Fox, and Sylvie Thiébaux, pp. 176–183. AAAI Press, (2007).
[12] Paul Höft, David Speck, and Jendrik Seipp. Code and data for the ECAI 2023 paper "Sensitivity Analysis for Saturated Post-hoc Optimization in Classical Planning". https://doi.org/10.5281/zenodo.8169126, 2023.
[13] Erez Karpas and Carmel Domshlak, 'Cost-optimal planning with landmarks', in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, ed., Craig Boutilier, pp. 1728–1733. AAAI Press, (2009).
[14] Erez Karpas, Michael Katz, and Shaul Markovitch, 'When optimal is just not good enough: Learning fast informative action cost partitionings', in *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)*, eds., Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp, and Malte Helmert, pp. 122–129. AAAI Press, (2011).
[15] Michael Katz and Carmel Domshlak, 'Optimal additive composition of abstraction-based admissible heuristics', in *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, eds., Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, pp. 174–181. AAAI Press, (2008).
[16] Michael Katz and Carmel Domshlak, 'Optimal admissible composition of abstraction heuristics', *Artificial Intelligence*, **174**(12–13), 767–798, (2010).
[17] Jonathan A. Kelner and Daniel A. Spielman, 'A randomized polynomial-time simplex algorithm for linear programming', in *Proceedings of the thirty-eighth annual ACM symposium on Theory of Computing*, pp. 51–61, (2006).
[18] Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp, 'From non-negative to general operator cost partitioning', in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, eds., Blai Bonet and Sven Koenig, pp. 3335–3341. AAAI Press, (2015).
[19] Florian Pommerening, Thomas Keller, Valentina Halasi, Jendrik Seipp, Silvan Sievers, and Malte Helmert, 'Dantzig-Wolfe decomposition for cost partitioning', in *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, eds., Robert P. Goldman, Susanne Biundo, and Michael Katz, pp. 271–280. AAAI Press, (2021).
[20] Florian Pommerening, Gabriele Röger, and Malte Helmert, 'Getting the most out of pattern databases for classical planning', in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, ed., Francesca Rossi, pp. 2357–2364. AAAI Press, (2013).
[21] Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet, 'LP-based heuristics for cost-optimal planning', in *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, eds., Steve Chien, Alan Fern, Wheeler Ruml, and Minh Do, pp. 226–234. AAAI Press, (2014).
[22] Jendrik Seipp, 'Online saturated cost partitioning for classical planning', in *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, eds., Robert P. Goldman, Susanne Biundo, and Michael Katz, pp. 317–321. AAAI Press, (2021).
[23] Jendrik Seipp and Malte Helmert, 'Counterexample-guided Cartesian abstraction refinement for classical planning', *Journal of Artificial Intelligence Research*, **62**, 535–577, (2018).
[24] Jendrik Seipp, Thomas Keller, and Malte Helmert, 'Narrowing the gap between saturated and optimal cost partitioning for classical planning', in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, eds., Satinder Singh and Shaul Markovitch, pp. 3651–3657. AAAI Press, (2017).
[25] Jendrik Seipp, Thomas Keller, and Malte Helmert, 'Saturated cost partitioning for optimal classical planning', *Journal of Artificial Intelligence Research*, **67**, 129–167, (2020).
[26] Jendrik Seipp, Thomas Keller, and Malte Helmert, 'Saturated post-hoc optimization for classical planning', in *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, eds., Kevin Leyton-Brown and Mausam, pp. 11947–11953. AAAI Press, (2021).
[27] Jendrik Seipp, Florian Pommerening, and Malte Helmert, 'New optimization functions for potential heuristics', in *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, eds., Ronen Brafman, Carmel Domshlak, Patrik Haslum, and Shlomo Zilberstein, pp. 193–201. AAAI Press, (2015).
[28] Daniel A. Spielman and Shang-Hua Teng, 'Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time', *Journal of the ACM*, **51**(3), (2004).
[29] Paul R. Thie and Gerard. E. Keough, *An introduction to linear programming and game theory*, John Wiley & Sons, 3rd edn., 2008.
[30] Fan Yang, Joseph Culberson, Robert Holte, Uzi Zahavi, and Ariel Felner, 'A general theory of additive state space abstractions', *Journal of Artificial Intelligence Research*, **32**, 631–662, (2008).