Dantzig-Wolfe Decomposition for Cost Partitioning

Florian Pommerening,[1] **Thomas Keller,**[1] **Valentina Halasi,**
Jendrik Seipp,[1,2] **Silvan Sievers,**[1] **Malte Helmert**[1]
[1] University of Basel, Switzerland
[2] Linköping University, Sweden
florian.pommerening@unibas.ch, tho.keller@unibas.ch, mail@valentinahalasi.com
jendrik.seipp@liu.se, silvan.sievers@unibas.ch, malte.helmert@unibas.ch

Abstract

Optimal cost partitioning can produce high quality heuristic estimates even from small abstractions. It can be computed with a linear program (LP) but the size of this LP often makes this impractical. Recent work used Lagrangian decomposition to speed up the computation. Here we use a different decomposition technique called Dantzig-Wolfe decomposition to tackle the problem. This gives new insights into optimal cost partitioning and has several advantages over Lagrangian decomposition: our method detects when a cost partition is optimal; it can deal with general cost functions; and it does not consider abstractions in the linear program that do not contribute to the heuristic value. We also show the advantage of the method empirically and investigate several improvements that are useful for all cost partitioning methods.

Introduction

The goal of optimal classical planning is to find a cheapest plan in a large factored state space. Planning tasks are commonly solved with heuristic search methods like A* search (Hart, Nilsson, and Raphael 1968). This requires an admissible heuristic function that maps states to lower bounds of their true goal distance. Cost partitioning (Katz and Domshlak 2010) is a way to combine multiple admissible heuristics into a single admissible estimate that can be higher than the maximum of its components. It works by partitioning the cost function of the problem and computing each heuristic function under a fraction of the cost.

Abstraction heuristics map the state space of a planning problem to a smaller abstract state space that can be fully explored. A single small abstraction is not very accurate but cost partitioning can combine many of them into a well-informed heuristic. The optimal way of partitioning the costs can be computed with a linear program (LP) in time polynomial in the size of all abstractions (Katz and Domshlak 2010). In practice, this LP is often too large to be useful.

Previous work used Lagrangian decomposition (Pommerening et al. 2019) to split up the large LP and iteratively compute suboptimal cost partitions that converge to an optimal one. We use a different technique called Dantzig-Wolfe decomposition (Dantzig and Wolfe 1960) which also computes a sequence of suboptimal cost partitions that converge to an optimum. Both methods present a new interpretation of cost partitioning. Pommerening et al. interpret Lagrangian multipliers as cost functions and the gradients used to update them as the number of times operators are used in a shortest plan. With Dantzig-Wolfe decomposition the problem decomposes into a master problem and several pricing problems and we interpret these LPs in terms of known planning concepts. On linear programs Dantzig-Wolfe decomposition is exactly dual to Benders decomposition (Benders 1962) so we also interpret the duals of master and pricing problems. In fact, our implementation is based on these dual problems, so it is closer to a Benders decomposition. We still use the view of Dantzig-Wolfe decomposition for most of the discussion as it directly works on the cost partitioning LP. Both views offer new insights into cost-partitioned heuristics.

Our method has several advantages over the one used by Pommerening et al.. Most importantly, it has a clear stopping condition, detecting whether an optimal cost partition is reached, while they use the subgradient method, which might not stop once the optimum is reached. Our method also supports general cost functions which can improve cost partitioning significantly (Pommerening et al. 2015) while theirs handles only non-negative ones. Finally, all known cost partitioning methods struggle with large collections of abstractions like the set of projections to at most three variables. We look at ways to mitigate this issue within the column generation used in Dantzig-Wolfe decomposition.

There is also a disadvantage compared to the method by Pommerening et al.: they replace all LPs with shortest path problems and compute them without an LP solver. Our algorithm relies on LP solvers for all involved LPs. We discuss the meaning of pricing problems but leave a more efficient implementation as an interesting algorithmic challenge.

In the remainder of this paper, we first introduce some notation for planning tasks and cost partitioning. We then explain Dantzig-Wolfe decomposition, first for general LPs and then specifically applied to the LP computing an optimal cost partition over abstraction heuristics. We then consider optional extensions of the basic method. In an experimental evaluation we show the use of these extensions and the advantage of Dantzig-Wolfe decomposition over computing the original LP or Lagrangian decomposition.

Background

We start with basic definitions and notation for cost partitioning and then explain Dantzig-Wolfe decomposition.

Planning

A *transition system* is a tuple $\langle \mathcal{S}, L, T, I, G, \mathit{cost} \rangle$, where $\mathcal{S}$ is a finite set of *states*, L a finite set of labels, each *transition*, $s \xrightarrow{\ell} t \in T \subseteq \mathcal{S} \times L \times \mathcal{S}$ connects two states and is annotated with a label. We are looking for a path following transitions from the *initial state* $I \in \mathcal{S}$ to some goal state s^* from the *set of goal states* $G \subseteq \mathcal{S}$. Each label is associated with a *cost* given by $\mathit{cost} : L \to \mathbb{R}$. The cost of a path is the sum of its label costs. The *perfect heuristic* maps each state s to the minimal cost of a path from s to some goal state, to ∞ if no such path exists, or to $-\infty$ if there is no minimal cost because paths with arbitrarily low cost exist (if a path to a goal state contains a cycle with negative total cost).

Planning tasks are compact, factored representations of transition systems. They use variable assignments over a finite set of *variables* as states. *Operators* are used as labels and can have preconditions and effects on variables (the former defining which states have an outgoing transition with this operator and the latter where this transition ends). We use the term "operator" instead of label when we want to emphasize that we are talking about the transition system of a planning task but otherwise will mostly talk about transition systems in general. Planning tasks can be projected to a subset of their variables (a *pattern*). The result of such a projection is a transition system (the *abstract state space*) that is usually much smaller than the one induced by the original task (the *original state space*). Projections preserve some structure of the original state space: there is a transition $\alpha(s) \xrightarrow{\ell} \alpha(s')$ in the abstract state space for all transitions $s \xrightarrow{\ell} s'$ in the original state space. This means that every plan in the original state space corresponds to a plan for the abstract state space and thus, the perfect heuristic for the abstract state space (h^α) cannot overestimate the perfect heuristic in the original state space (h^*). This makes h^α an admissible heuristic in the original state space.

Consider a set of abstractions $A = \{\alpha_1, \dots, \alpha_n\}$. Each heuristic h^{α_i} provides an admissible estimate but for A* we have to combine them in a way that maintains the admissibility. Cost partitioning (Katz and Domshlak 2010) achieves this by evaluating each heuristic h^{α_i} under a different cost function cost_i. We write $h^{\alpha_i}(s, \mathit{cost}_i)$ for the cost of a cheapest path in abstraction α_i but using the cost function cost_i instead of cost. The sum of these values ($\sum_{i=1}^n h^{\alpha_i}(s, \mathit{cost}_i)$) is admissible if the cost functions cost_i form a partition of the original cost function cost, i.e., if they satisfy $\sum_{i=1}^n \mathit{cost}_i(\ell) \leq \mathit{cost}(\ell)$ for each label $\ell \in L$. Katz and Domshlak showed that an optimal way of partitioning the costs can be computed in time polynomial in the size of the abstract state spaces. This is done with a linear program (LP) where variables encode the local cost functions ($\mathit{cost}_i(\ell)$ for each i and ℓ) and the distance of each abstract state from the current state under cost_i. Despite its polynomial size this LP is often prohibitively large in practice.

Dantzig-Wolfe Decomposition

Dantzig-Wolfe decomposition (Dantzig and Wolfe 1960) is a technique from operations research to solve large programs with a specific structure. Before we apply it to cost partitioning, we describe the technique for a general LP. The decomposition consists of two steps: rewriting the LP into an alternative form and using column generation (Ford and Fulkerson 1958) on the rewritten LP. For a more detailed introduction we refer to Desrosiers and Lübbecke (2005).

Rewriting the LP Consider the following LP:

$$\text{Maximize} \sum_{1 \leq i \leq n} o_i^\top x_i \text{ subject to}$$

$$\sum_{1 \leq i \leq n} A_i x_i \leq b_0 \tag{1}$$

$$B_i x_i \leq b_i \qquad \text{for all } 1 \leq i \leq n \tag{2}$$

where all x_i are vectors of LP variables, all o_i and b_i are coefficient vectors, and all A_i and B_i are coefficient matrices. Constraint (1) is called the *complicating constraint*, while the constraints (2) are called *subproblem constraints*. The LP would decompose into subproblems that could be maximized independently if the complicating constraint were not present (every x_i only occurs in subproblem constraint i and the complicating constraint).

The solutions of subproblem constraint i form a polyhedron $P_i = \{x_i \mid B_i x_i \leq b_i\}$ so the constraint can be written as $x_i \in P_i$. The main result underlying the Dantzig-Wolfe decomposition is Minkowski's Theorem (Schrijver 1998) stating that every polyhedron is generated by finitely many *rays* and *extreme points*. That is, there is a set of rays r_{ij} and extreme points p_{ik} such that $x_i \in P_i$ iff $x_i = \sum_k \lambda_{ik} p_{ik} + \sum_j \delta_{ij} r_{ij}$ for some coefficients $\lambda_{ik} \geq 0$ satisfying $\sum_k \lambda_{ik} = 1$ (convex combination of extreme points) and some coefficients $\delta_{ij} \geq 0$ (linear combination of rays). This representation is called the *Minkowski sum* of P_i. We could use it to rewrite the constraint $x_i \in P_i$ if we had all rays r_{ij} and extreme points p_{ik} generating P_i.

At this point we deviate from the typical explanation of Dantzig-Wolfe decomposition to simplify things. In the LP we will consider later, all b_i with $1 \leq i \leq n$ are 0. This means that $P_i = \{x_i \mid B_i x_i \leq 0\}$ is a special case of a general polyhedron called a *polyhedral cone*. It can have at most one extreme point (at 0) and its Minkowski sum simplifies to consider only the rays: $x_i \in P_i$ iff $x_i = \sum_j \delta_{ij} r_{ij}$ for some coefficients $\delta_{ij} \geq 0$. Let us use this to eliminate the variables x_i and express the whole LP in terms of δ_{ij}.

$$\text{Maximize} \sum_{1 \leq i \leq n} \sum_j o_i^\top \delta_{ij} r_{ij} \text{ subject to}$$

$$\sum_{1 \leq i \leq n} \sum_j A_i \delta_{ij} r_{ij} \leq b_0$$

$$\delta_{ij} \geq 0 \qquad \text{for all } i \text{ and } j$$

This representation is called the *master problem* and is equivalent to the original problem. It has fewer constraints but typically the number of rays is so high that even constructing the master problem is infeasible.

Column Generation An optimal solution to the master problem typically leaves most variables at 0. If we knew these variables in advance, we could remove them without affecting the objective value. In absence of this knowledge, we can generate the variables (columns in the coefficient matrix) iteratively until the solution is guaranteed to be optimal.

To do so, we consider the *restricted master problem (RMP)* that is like the master problem but only uses a subset of its columns. Leaving out variables means that the RMP can underestimate the objective value of the master problem. According to the duality theorem (Schrijver 1998), those ignored columns correspond to constraints in the dual RMP and adding a column is the same as adding a dual constraint. In fact, the method can also be seen as generating constraints in the dual RMP rather than columns in the primal RMP. This is called Benders decomposition (Benders 1962) and is exactly dual to Dantzig-Wolfe decomposition on LPs.

Finding a dual constraint that is currently violated is called the *pricing problem*. Formally, the dual of the master problem from the previous section is

$$\text{Minimize } b_0^\top y \text{ subject to}$$
$$\begin{aligned} (A_i r_{ij})^\top y &\geq o_i^\top r_{ij} && \text{for all } i \text{ and } j \\ y &\geq 0 \end{aligned}$$

We can look for the most violated dual constraint in each subproblem separately and then compare them to find the most violated constraint or just add all of them to the RMP. Assume that the vector y is an optimal solution to the dual RMP, i.e., the LP above with all constraints generated so far. We want to find a violated constraint, i.e., a new ray r_{ij} of the polyhedron P_i for which $(A_i r_{ij})^\top y - o_i^\top r_{ij} < 0$. The most violated constraint minimizes the left-hand side. Using the fact that r_{ij} is a ray of $P_i = \{x_i \mid B_i x_i \leq 0\}$, we can write down the pricing problem $P_i(y)$:

$$\text{Minimize } (A_i x_i)^\top y - o_i^\top x_i \text{ subject to}$$
$$B_i x_i \leq 0$$

Note that this pricing problem is small relative to the RMP and the original problem: it only uses variables and constraints of a single subproblem of our original problem.

To solve the original problem with Dantzig-Wolfe decomposition, we thus perform the following steps:

1. Compute an optimal solution y to the dual RMP (initially without constraints).
2. For each i, optimize the pricing problem $P_i(y)$. LP solvers will either produce a ray with negative objective value if the problem is unbounded or an extreme point if it is bounded.
   (a) A ray r_{ij} represents a violated constraint. Add the constraint $(A_i r_{ij})^\top y \geq o_i^\top r_{ij}$ to the dual RMP (or the column for r_{ij} to the primal RMP).
   (b) The only possible extreme point in our case is 0 (recall that P_i is a polyhedral cone). In this case no constraint for this subproblem is violated in the dual RMP.
3. If no pricing problem generates a ray, y is feasible in the dual master problem, and the corresponding primal solution is optimal for our original problem. Otherwise repeat from step 1.

Application to Cost Partitioning

We now apply Dantzig-Wolfe decomposition to the optimal cost partitioning over abstraction heuristics. Let $A = \{\alpha_1, \dots, \alpha_n\}$ be abstractions of a common original problem with state space $\langle \mathcal{S}, L, T, I, G, \mathit{cost}\rangle$. The abstract state space of α_i is $\langle \mathcal{S}_i, L, T_i, I_i, G_i, \mathit{cost}\rangle$. It is important that all transitions $s \xrightarrow{\ell} t \in T_i$ are *alive*, i.e., that s is reachable from I_i and an abstract goal state is reachable from t. We assume that all other transitions are removed from T_i. The optimal cost partitioning over A in the initial state[1] can be computed as follows:

$$\text{Maximize } \sum_{1 \leq i \leq n} h_i \text{ subject to}$$
$$\begin{aligned} \sum_{1 \leq i \leq n} c_{i\ell} &\leq \mathit{cost}(\ell) && \text{for all } \ell \in L && (3) \\ d_{iI_i} &= 0 && \text{for all } i && (4) \\ d_{it} &\leq d_{is} + c_{i\ell} && \text{for all } i \text{ and } s \xrightarrow{\ell} t \in T_i && (5) \\ h_i &\leq d_{is^*} && \text{for all } i \text{ and } s^* \in G_i && (6) \end{aligned}$$

The cost partitioning constraint (3) is the complicating constraint as it involves variables from all abstractions. Constraints (4)–(6) form one subproblem constraint per abstraction. As before, the subproblems would be independent without the complicating constraint. To match this with (1)–(2), matrix A_i has one row for each $\ell \in L$ with 1 in the column for $c_{i\ell}$ and 0 everywhere else; the vector b_0 corresponds to the cost function *cost*, while all other bound vectors b_i are 0. The objective coefficients o_i are 1 in the column for h_i and 0 everywhere else. Using these definitions, we can now construct the RMP and the pricing problems.

The (Relaxed) Master Problem

The RMP is based on rays of the subproblems. In our case of a polyhedral cone, the set of rays and the feasible solution space are identical because every solution can be scaled up or down, i.e., rays are specific solutions to the subproblem constraints. Each ray r_{ij} consists of values h_{ij}, $c_{ij\ell}$ for all labels $\ell \in L_i$, and values d_{ijs} for each abstract state $s \in \mathcal{S}_i$. In the master problem j ranges over all rays in the Minkowski sum of P_i, while in the RMP it ranges over all rays generated for subproblem i so far.

$$\text{Maximize } \sum_{1 \leq i \leq n} \sum_j h_{ij} \delta_{ij} \text{ subject to}$$
$$\begin{aligned} \sum_{1 \leq i \leq n} \sum_j \delta_{ij} c_{ij\ell} &\leq \mathit{cost}(\ell) && \text{for all } \ell \in L \\ \delta_{ij} &\geq 0 && \text{for all } i \text{ and } j \end{aligned}$$

Note that the coefficients of all d_{ijs} are 0 so we can treat rays as a cost function (encoded in the values $c_{ij\ell}$) and a value h_{ij}. We tentatively interpret the h_{ij} as the heuristic value under that cost function (and will later see this confirmed). The cost functions $\mathit{cost}_i(\ell) = \sum_j \delta_{ij} c_{ij\ell}$ form a cost partition and the heuristic value of abstraction i under cost_i is

[1]Using the initial state here simplifies the presentation. The heuristic can be computed for other states s by replacing I with s and I_i with the abstract state representing s in abstraction i.

$\sum_j h_{ij}\delta_{ij}$. Since this is true for any choice of $\delta_{ij} \geq 0$, the LP computes an optimal choice of δ_{ij} given the available cost functions encoded in $c_{ij\ell}$. This gives a first interpretation of the method: the pricing problems generate *candidate cost functions* together with the heuristic values achieved under them; the RMP then "mixes" the candidates as a linear combination that satisfies the cost partitioning constraint. The resulting cost partition is optimal only for the current set of candidates, so the pricing problems add more candidates to achieve a better global value.

The Dual (Relaxed) Master Problem

As the pricing problems are parameterized with a solution of the dual RMP it is worth looking at this LP as well. The view of generating constraints for this dual RMP rather than columns for the primal corresponds to Benders decomposition. While it is known that optimal cost partitioning is dual to operator-counting heuristics (Pommerening et al. 2015), we are interested in *what kind* of operator-counting constraints the heuristic contains. The dual RMP is:

$$\begin{aligned} &\text{Minimize} \sum_{\ell \in L} \mathit{cost}(\ell)\, y_\ell \text{ subject to} \\ &\sum_{\ell \in L} c_{ij\ell} y_\ell \geq h_{ij} \qquad && \text{for all } i \text{ and } j \\ &y_\ell \geq 0 && \text{for all } \ell \in L \end{aligned}$$

Let cost_{ij} be the cost function encoded by the values $c_{ij\ell}$. Assume that h_{ij} is the heuristic value of abstraction i under cost_{ij} (as we suspected before). Then the dual RMP is an operator-counting heuristic with constraints similar to post-hoc-optimization constraints (Pommerening et al. 2014), which are defined as $\sum_{\ell \in L} \mathit{cost}(\ell)\, y_\ell \geq h(s, \mathit{cost})$. The constraints here have the form $\sum_{\ell \in L} \mathit{cost}'(\ell)\, y_\ell \geq h(s, \mathit{cost}')$ for an alternative cost function cost', which is a valid operator counting constraint for any choice of cost' (Thüring 2019). The resulting constraint is still an operator-counting constraint, so it can be used in the operator-counting framework. Other operator-counting constraints could be added to the RMP to strengthen the heuristic but we do not pursue this option further to focus on the core problem.

The Pricing Problems

The pricing problem $P_i(y)$ for subproblem i depends on an optimal solution y of the dual RMP. If we plug in the specific matrix A_i (only 1 for row ℓ, column $c_{i\ell}$ and otherwise 0) and vector o_i (only 1 for h_i), we see that the pricing problem has the objective $(A_i x_i)^\top y - o_i^\top x_i = \sum_{\ell \in L} y_\ell\, c_{i\ell} - h_i$. The constraints are just the subproblem constraints for subproblem i, so the pricing problem $P_i(y)$ is

$$\begin{aligned} &\text{Minimize} \sum_{\ell \in L} y_\ell\, c_{i\ell} - h_i \text{ subject to} \\ &d_{iI_i} = 0 \\ &d_{it} \leq d_{is} + c_{i\ell} \qquad && \text{for all } s \xrightarrow{\ell} t \in T_i \\ &h_i \leq d_{is^*} && \text{for all } s^* \in G_i \end{aligned}$$

The constraints closely resemble a shortest path problem for abstraction α_i but the objective is different. Instead of maximizing the heuristic value (resp. minimizing $-h_i$) under a given cost function, we can choose the cost function freely, but get a penalty in the objective for the costs we use (weighted by the parameter y).

Another way to look at this is to consider a cost function cost_i. The constraints can be satisfied with $c_{i\ell} = \mathit{cost}_i(\ell)$ for any $h_i \in \mathbb{R}$ with $h_i \leq h^{\alpha_i}(I, \mathit{cost}_i)$. The values of the variables d_{is} do not matter here because they are ignored in the RMP. If we define $\mathit{cost}_i(y) = \sum_{\ell \in L} \mathit{cost}_i(\ell)\, y_\ell$, we can rewrite the pricing problem as

$$\begin{gathered} \text{Minimize } \mathit{cost}_i(y) - h_i \text{ subject to} \\ h_i \leq h^{\alpha_i}(I, \mathit{cost}_i) \\ \mathit{cost}_i : L \to \mathbb{R}, h_i \in \mathbb{R} \end{gathered}$$

Note that that the pricing problem generates a new column if its objective value is negative. If this is the case for some cost function cost_i and value h_i, it is also true for $h'_i = h^{\alpha_i}(I, \mathit{cost}_i)$ and cost'_i defined as the saturated cost function of cost_i (Seipp, Keller, and Helmert 2020) where the cost of each label is reduced as much as possible without affecting the goal distance of any state. In our experiments, we always do this step to achieve tighter constraints.

To better understand the pricing problem, we now look at its dual.

The Dual Pricing Problems

The dual of the pricing problem $P_i(y)$ can be written as

$$\begin{aligned} &\text{Maximize } 0 \text{ subject to} \\ &\sum_{\substack{t \text{ labeled} \\ \text{with } \ell}} f_t = y_\ell \qquad && \text{for all } \ell \in L \qquad (7) \\ &\sum_{t \text{ leaves } s} f_t + g_s[s \in G_i] = \sum_{t \text{ enters } s} f_t && \text{for all } s \neq I_i \qquad (8) \\ &\sum_{s^* \in G_i} g_{s^*} = 1 && \qquad (9) \\ &f_t \geq 0 \text{ for all } t \in T_i \text{ and } g_{s^*} \geq 0 \text{ for all } s^* \in G_i && \qquad (10) \end{aligned}$$

Constraints (8)–(10) describe a network flow problem in the abstraction where f_t describes the amount of flow along a transition. Constraint (7) requires that the flow is *consistent* with y, i.e., that the aggregated flow for a label ℓ matches y_ℓ. If such a flow exists, the objective value is 0 so the primal pricing problem also has a value of 0 and does not generate a column. In case no such flow exists, the dual is infeasible and the primal is unbounded and generates a ray in the RMP.

This agrees with the view of the dual RMP as an operator-counting LP: the dual solution y is an *operator count* mapping each operator to a number of uses. If an abstraction does not have a consistent flow, it generates a new operator-counting constraint demonstrating that the cost of a cheapest flow under a cost function cost_i must be lower than $\mathit{cost}_i(y)$.

Extensions

In this section, we discuss several optional extensions to the basic algorithm outlined above.

Combining Labels

In abstractions (particularly in small ones), some labels can be equivalent. For example, think of a Blocksworld task projected to the single variable indicating whether a certain block b is clear. Stacking any block on top of b leads to an abstract transition from *clear* to *not clear*. In the abstraction all of these operators will label parallel transitions.

We say two labels $\ell, \ell' \in L$ are equivalent in an abstraction with transitions T if they only label parallel transitions: $\ell \sim \ell'$ iff $T(\ell) = T(\ell')$ where $T(\ell) = \{\langle s, t\rangle \mid s \xrightarrow{\ell} t \in T\}$. Let $[\ell]$ be the equivalence class of label ℓ under $\sim$. The abstraction can be compactly represented with labels $\{[\ell] \mid \ell \in L\}$ and transitions $\{s \xrightarrow{[\ell]} t \mid s \xrightarrow{\ell} t \in T\}$, i.e., by combining all equivalent labels into a single new label.

With cost partitioning, the original label costs now have to be partitioned among local label cost functions that do not share labels. Implicitly, we compute a cost partition that assigns the same cost to all equivalent labels. A cost partition that assigns a higher cost to some label in an equivalence class cannot yield a higher heuristic value, as the cheapest path between two states is determined by the minimum cost of any label on a transition between them. The cost partitioning constraint for label ℓ changes to partition the cost of ℓ among its equivalence classes from all abstractions:

$$\sum_{1 \le i \le n} c_{[\ell]_i} \le \mathit{cost}(\ell)$$

All other constraints are changed to use the compact form of transitions defined above and the label cost variables $c_{[\ell]_i}$ instead of $c_{i\ell}$. While the size of the RMP is not affected by this, the size of the pricing problems is reduced as they are now phrased in terms of the compact representation. We want to emphasize the large reduction this simple change can bring. While the number of operators in a planning task is often in the thousands, in a projection to a binary variable, at most seven different equivalence classes are possible.

The idea to combine equivalent labels is not new. The merge-and-shrink framework stores abstractions in this compact form (Sievers 2018) and even reduces the amount of labels further with *label reduction*. Sievers et al. (2020) define *extended label cost partitioning* for merge-and-shrink abstractions which matches our definition. We note that this way of computing compact transition systems and cost partitions over them is beneficial not only for merge-and-shrink or Dantzig-Wolfe decomposition, but for all cost partitioning methods over abstractions.

A small extension is to not explicitly represent a label for the equivalence class of irrelevant labels, i.e., labels that only induce self-looping transitions at all states. These cannot contribute a positive cost to optimal solutions and cannot take a negative cost without resulting in a heuristic value of $-\infty$. We therefore replace their LP variables by the constant 0.

Restricting the Considered Patterns

Pommerening, Röger, and Helmert (2013) define the notion of *interesting* patterns for non-negative cost partitioning. It is based on the *causal graph* of a given planning task Π, which is a directed graph with the variables of Π as nodes. It contains a *precondition edge* from v to w if there is an operator with a precondition on v and an effect on w, and it contains an *effect edge* from v to w if there is an operator affecting both v and w. A pattern P is interesting for non-negative cost partitioning if the causal graph restricted to P is weakly connected and contains a path using precondition edges that leads from each variable in P to a a variable mentioned in the goal of Π. Pommerening, Röger, and Helmert show that a projection to an uninteresting pattern can be replaced by one or more projections to smaller patterns without changing the value of an optimal non-negative cost partitioning.

Unfortunately, this definition of interesting patterns is too strict for cost partitioning with general cost functions. For example, Pommerening et al. (2015) show an example where a projection to a single non-goal variable is relevant (Figure 1 in their paper). Therefore, all previous work on general cost partitioning does not exclude any projections. However, there are clearly patterns which we can identify as redundant for general cost partitioning.

Definition 1. *A pattern P is* redundant for general cost partitioning *in a task with causal graph G if*

- *G restricted to P is not weakly connected, or*
- *P contains a variable v such that G has no directed path along precondition edges from v to a goal variable.*

The difference between this definition and the one for non-negative costs is in the second condition. We consider the full causal graph G there, not its restriction to P. Note that variables that satisfy this second condition can be removed from the planning task in a preprocessing step and the definition simplifies to the first condition if this is done.

We can show that projections to redundant patterns can be replaced by projections to smaller patterns without affecting the heuristic value. For the full proofs, we refer to a technical report (Pommerening et al. 2021b). The proof idea in the first case is to replace P by the connected components of G restricted to P. No operator can affect more than one connected component and operators not affecting a component induce self-looping transitions on all states. In the second case P can be replaced by $P' = P \setminus \{v\}$. In an optimal cost partition, we have to set the cost of all operators (in all abstractions) that depend directly or indirectly on v to 0. This cannot affect the heuristic value as no path to the goal can depend on v. The difficult part in the proofs is to show that changing the cost partitions does not introduce a cycle of negative cost.

We conjecture that this definition is the most restrictive, i.e., any stricter definition considering just the patterns and the causal graph would exclude useful patterns.

Incrementally Extending the Set of Subproblems

Corrêa and Pommerening (2019) show that encoding the optimal heuristic usually requires considering a large number of small projections but only a small number of larger projections. (They consider potential heuristics which partition costs over projection heuristics.) Their result implies that a low number of abstractions is often sufficient to find plans that are plans for larger abstractions. If this extends to our setting, a large fraction of the pricing problems will often accept the dual RMP solution as a consistent network flow and will not generate a new column. We can use this by incrementally extending the set of subproblems.

Instead of querying all pricing problems in each iteration, we can query just a subset of them. When none of them generates a new column, we extend the set and repeat the process. We use a simple strategy to extend the set: we iterate over all candidate patterns ordered by size, generate the abstraction and pricing problem and check if it generates a new column. If so, we add it and break out of the loop. Observe that even this simple strategy can pay off because it does not consider all pricing problems in every iteration and it has fewer abstractions in memory at the same time. We will explore the trade-off between solving fewer pricing problems per iteration and requiring more iterations empirically. We leave more advanced techniques for future work.

Note that there is a subtle issue with this incremental process: we assume that our abstractions only contain alive transitions and explicitly remove other transitions. If an operator induces no alive transitions in an abstraction, it is considered *dead* and cannot be used in the solution. We fix operator-counting variables of such operators to 0 in the dual RMP. When constructing the set of abstractions incrementally, we do not know the set of dead operators in advance, so constructing an abstraction for the first time might show that an operator is dead. We treat this as if the pricing problem had generated a column and add it to the considered set.

Using the Heuristic in Search

The naive way of using the Dantzig-Wolfe method within search is to restart the process from scratch in every state. As this is prohibitively expensive, we instead keep the RMP and every pricing problems that is ever generated and update them when we evaluate a new state. We interpret the generated columns as pairs $\langle h, \mathit{cost} \rangle$ where *cost* is a saturated cost function and $h = h^{\alpha}(s, \mathit{cost})$ is the heuristic value for the current state under this cost. If the search moves from state s to a new state s', we compute $h' = h^{\alpha}(s', \mathit{cost})$ and update all generated columns by substituting h with h'. In the primal RMP, this updates the objective, in the dual RMP it updates the bounds of the post-hoc optimization constraints.

In addition, we have to recompute the set of dead operators in every state, because the notion of an operator not inducing alive transitions in an abstraction depends on which abstract states are reachable from the current state.

Once the RMP and the pricing problems are updated, we continue with the heuristic computation as described above by querying all existing subproblems for new columns and by iteratively adding all candidate patterns that create a new column for s' to the set of subproblems. The resulting heuristic value is guaranteed to match the optimal cost partitioning value for all patterns. In the following, we call this the KEEPALL strategy.

Relaxing the Heuristic

Computing an optimal cost partition in each state of a search might be too expensive. We consider three relaxations of the heuristic that all compute admissible approximations:

KEEPPATTERNS When incrementally extending the set of subproblems, we extend the set until no new columns are generated in the initial state, but do not extend it in any subsequent state. We still update the bounds of constraints and add columns from all considered subproblems. This selects a set $A' \subseteq A$ of abstractions that is sufficient to compute an optimal cost partition over all abstractions A in the initial state. In subsequent states, the computed cost partition is optimal for A' but not necessarily for A.

KEEPRMP We neither extend the set of subproblems nor do we evaluate pricing problems in states other than the initial state, i.e., we stick to the columns in the RMP after the computation of the initial state. This requires only a single LP evaluation of the RMP in each state and the memory used for the pricing problems can be reused. The resulting heuristic value is that of a suboptimal cost partition over the abstractions used in the initial state.

KEEPCP After the computation for the initial state, we extract the cost functions $\mathit{cost}_i(\ell) = \sum_j \delta_{ij} c_{ij\ell}$ from the RMP solution δ. We compute the abstract goal distances in abstraction α_i under cost_i and store the distances. The resulting heuristic can be seen as additive pattern databases (Culberson and Schaeffer 1998; Edelkamp 2001) or potential heuristics (Pommerening et al. 2015).

Each method is a relaxation of the methods above it, trading off heuristic quality for savings on computation time and memory usage. We will explore this in our experiments.

Experiments

We implemented Dantzig-Wolfe decomposition in Fast Downward 20.06 (Helmert 2006) with CPLEX 12.10 as an LP solver and used Downward Lab (Seipp et al. 2017) to run experiments. Our benchmark set consists of all 1827 tasks without conditional effects from the optimal sequential tracks of the International Planning Competitions 1998–2018. We limit memory to 3.5 GiB. Time is limited to 5 minutes in experiments evaluating only the initial state and 30 minutes otherwise. All benchmarks, code and experiment data are published online (Pommerening et al. 2021a).

In all experiments, we construct the dual rather than the primal RMP, generating constraints rather than columns. This implementation as Benders decomposition fits better into the existing implementation of operator-counting. We use general cost partitioning unless stated differently. All experiments are run with two sets of abstractions: all projections to up to two (SYS2) and up to three (SYS3) variables.

We start our empirical evaluation with a configuration DW_{base} that uses no additional extensions. This configuration computes an optimal cost partition for the initial state in 1018 out of 1827 tasks for SYS2 and in 321 tasks for SYS3 (see Table 1). The configuration $\text{OCP-LP}_{\text{base}}$ computes the same heuristic as a monolithic LP. Comparing both clearly shows the benefit of the decomposition which solves 305 additional tasks for SYS2 and 87 for SYS3. As there are still many tasks where optimally partitioning the costs is not feasible we evaluate if our proposed extensions can help.

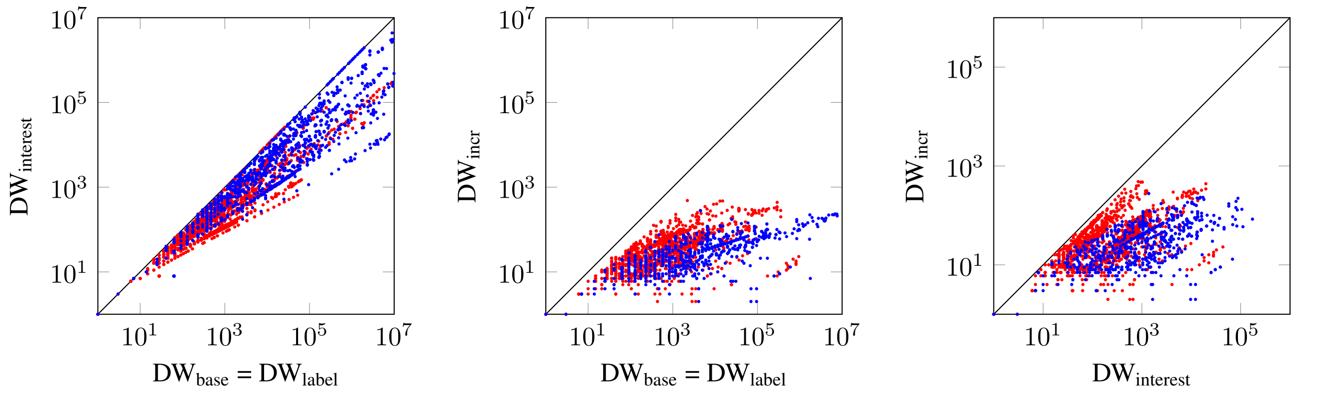

Figure 1: Number of considered patterns from Sys2 (red) and Sys3 (blue). We only consider instances where data for both configurations is available.

	labels combined	interesting patterns	incremental extensions	Sys2	Sys3
OCP-LP$_{\mathrm{base}}$	-	-	n/a	713	234
OCP-LP$_{\mathrm{label}}$	+	-	n/a	1042	281
OCP-LP$_{\mathrm{interest}}$	+	+	n/a	1161	379
DW$_{\mathrm{base}}$	-	-	-	1018	321
DW$_{\mathrm{label}}$	+	-	-	1167	358
DW$_{\mathrm{interest}}$	+	+	-	1456	561
DW$_{\mathrm{incr}}$	+	+	+	**1461**	**772**

Table 1: Number of tasks for which the heuristic value for the initial state can be computed within the resource limits.

Basic Extensions

We first consider combining equivalent labels in configuration DW$_{\mathrm{label}}$. We see a strong impact in Table 1. The more compact representation of the transition systems requires significantly less memory allowing the method to compute a heuristic value in tasks where DW$_{\mathrm{base}}$ ran out of memory. Combining labels has an even larger impact on the monolithic LP (OCP-LP$_{\mathrm{label}}$) but the Dantzig-Wolfe method still comfortably outperforms this configuration.

We now keep labels combined but consider only projections to non-redundant patterns (DW$_{\mathrm{interest}}$). This shows an even stronger impact on the number of tasks where we can compute the heuristic with 289 additional initial heuristic values for Sys2 and 203 for Sys3. The first plot in Figure 1 shows that this is because DW$_{\mathrm{base}}$ and DW$_{\mathrm{label}}$ use up to 80 times as many patterns for the computation of the initial heuristic value as DW$_{\mathrm{interest}}$ for Sys2 and up to 450 times as many for Sys3. Again, this extension also has a positive impact on the monolithic LP (OCP-LP$_{\mathrm{interest}}$) but in this case, the effect on the Dantzig-Wolfe method is stronger.

The configuration DW$_{\mathrm{incr}}$ uses both extensions considered above but reduces the number of considered subproblems further by incrementally extending the set of subproblems starting from projections to goal variables and going up to Sys2/Sys3. The results in Table 1 indicate a small additional impact in Sys2 and a large effect in Sys3. The number of considered patterns (last two plots in Figure 1) explains this: DW$_{\mathrm{incr}}$ usually requires less than 100 of the up to 10^7 patterns to compute an optimal cost partition, and the number of patterns considered by DW$_{\mathrm{base}}$ and DW$_{\mathrm{label}}$ is up to 50000 times as high for Sys2 and up to 250000 for Sys3. This fits with the results by Corrêa and Pommerening (2019) discussed earlier. Considering fewer abstractions improves the memory usage of the algorithm dramatically. With DW$_{\mathrm{interest}}$ for Sys3, over 1000 tasks run out of memory, while none run out of memory in DW$_{\mathrm{incr}}$. We see this method trades time for memory in an overall favorable rate.

Overall, the experiments summarized in Table 1 show that all proposed extensions have a positive impact on Dantzig-Wolfe decomposition. Smaller differences for extensions that are added later do not necessarily mean smaller effects because of diminishing returns.

Comparison to Other OCP Methods

We now compare the strongest configuration from the previous section (DW$_{\mathrm{incr}}$) to the Lagrangian decomposition method (LG) by Pommerening et al. (2019) and the monolithic LP (OCP-LP$_{\mathrm{interest}}$). As the Lagrangian method cannot handle general cost functions, we restrict all methods to non-negative costs in this section. We also combine equivalent labels in LG and consider only patterns that are interesting for non-negative costs in all configurations.

Since Lagrangian decomposition does not detect when it has reached an optimum, we cannot compare the number of instances for which an optimal partition can be computed. However, the anytime nature is a strength of Lagrangian decomposition. As Dantzig-Wolfe also computes a sequence of cost partitions of increasing quality, we compare the approaches based on their heuristic quality as a function of time in the first plot of Figure 2. We seed the Lagrangian decomposition version with a saturated cost partitioning (Seipp, Keller, and Helmert 2020) which is optimized by hill-climbing for 100 seconds. This configuration achieved the highest heuristic estimates in previous work (Pommerening et al. 2019). We compute heuristic quality of

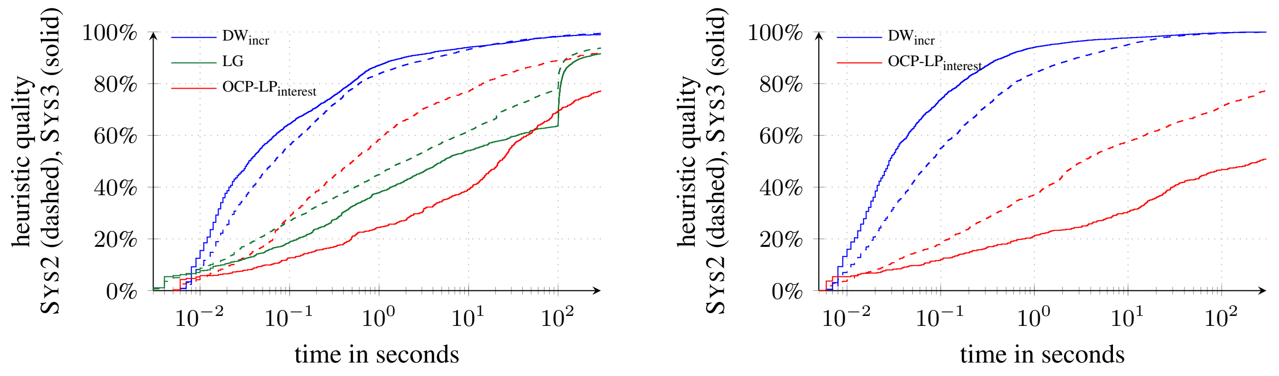

Figure 2: Heuristic quality for the initial state over time for non-negative (left) and general (right) cost partitioning methods.

a task at time t as the ratio of the heuristic value it would get if the heuristic computation were stopped after t seconds and the value of an optimal cost partition. The figure shows the average of these values at each time. Note that the plots for SYS2 and SYS3 are incomparable because their reference values are different.

For both pattern collections, we see that DW$_{incr}$ clearly outperforms the other methods, i.e., it finds higher heuristic values more quickly. The configuration OCP-LP$_{interest}$ can only have a quality of 0 or 1 for a task, so the fact that it has much lower values in the plot reflects that it takes much longer to compute the optimal values with a monolithic LP. This is even more pronounced when computing the general OCP (see the second plot of Figure 2). Here, DW$_{incr}$ achieves a heuristic quality of 1.0 for SYS2 and SYS3 after 300 seconds, while OCP-LP$_{interest}$ only reaches values of 0.77 (SYS2) and 0.51 (SYS3).

Usage as Heuristic

We proposed four ways of using the Dantzig-Wolfe heuristic in a search. In all cases, we first compute DW$_{incr}$ as described above. Depending on the strategy, we then remove the pricing problems (KEEPCP, KEEPRMP) and the RMP (KEEPCP) from memory. In every state we update the bounds of the remaining LPs, compute the set of dead operators and fix their operator counts to 0 as described above. The strategies span a range from expensive-to-compute but accurate to cheap-to-compute but less accurate. We now explore this trade-off.

Incrementally extending the set of subproblems is crucial for all strategies other than KEEPALL because they benefit from a small set of subproblems. However, we have seen that the incremental extension takes time, and might be too much to repeat in every state. We thus additionally test a configuration KEEPALLNOINCR that is based on DW$_{interest}$ instead of DW$_{incr}$. The coverage and per-domain coverage comparison of the results in Table 2 show that the incremental extension is indeed too expensive to perform in every step.

Table 2 also shows that performance increases the more we relax the heuristic. The relaxations do not seem to lose much accuracy and the expensive heuristic computation does not pay off in comparison. This is not surprising,

SYS2	(A)	(B)	(C)	(D)	(E)	coverage
(A) KEEPALL	–	10	8	7	10	693
(B) KEEPALLNOINCR	**26**	–	9	8	10	750
(C) KEEPPATTERNS	**36**	**27**	–	8	9	823
(D) KEEPRMP	**46**	**41**	**30**	–	12	890
(E) KEEPCP	**47**	**46**	**40**	**35**	–	**974**
SYS3	**(A)**	**(B)**	**(C)**	**(D)**	**(E)**	**coverage**
(A) KEEPALL	–	12	1	1	2	339
(B) KEEPALLNOINCR	**18**	–	5	2	3	362
(C) KEEPPATTERNS	**46**	**42**	–	1	7	568
(D) KEEPRMP	**54**	**50**	**35**	–	9	665
(E) KEEPCP	**57**	**54**	**35**	**23**	–	**698**

Table 2: Per-domain coverage comparison of different strategies. The entry in row r and column c shows the number of domains in which r solves more tasks than c. The maximum of entries (r, c) and (c, r) is highlighted.

as suboptimal cost partitioning techniques often outperform optimal ones. While the KEEPCP strategy does not achieve state-of-the-art performance, there are many options for improvement. The computation runs out of time in the initial state for 259 tasks. Due to the iterative nature of DW$_{incr}$, the process can be stopped before an optimal cost partition is computed to ensure that search always starts. The strategy also never exhausts time (only memory) after the search starts, which shows that search and heuristic evaluation are fast. Trading off time for improved heuristic quality should hence improve performance. Several enhancements that could be used here have been proposed for saturated cost partitioning, e.g., using multiple cost partitions (Seipp, Keller, and Helmert 2020), or using abstractions from a larger set as long as resources permit (Seipp 2019).

Future Work

Our method can be seen as Dantzig-Wolfe or Benders decomposition of a large LP. Many techniques from operations research tackle convergence issues in such decompositions and are directly applicable here (e.g., stabilization methods). In contrast to the typical use case in OR, we solve related versions of the original LP multiple times to compute heuristic values for each state encountered in the search. This is an interesting use case for such methods.

We see three additional areas for future work. Firstly, we focused mainly on computing a single heuristic value and only briefly explored the use of our method as a heuristic. Additional work is required for state-of-the-art performance. Secondly, the main effort in computing heuristic values stems from evaluating LPs. We could avoid some of it by precomputation. Finally, we might be able to avoid using an LP solver altogether if we were able to find efficient algorithms for two clearly defined problems. In the remainder of this section, we discuss these three areas in more detail.

Improving Usage as Heuristic

Currently, the strategies KEEPALL and KEEPPATTERNS keep adding constraints to the dual RMP in every state. Constraints for previous states are kept and updated with new bounds. Generated constraints can remain useful and cause fewer constraints to be generated for new states. While the number of constraints we add in this way is limited by the combined size of all skeletons it is easy to imagine cases where it becomes too large. To mitigate this, we should thus explore strategies to remove constraints. One option would be to track how often a column was useful and remove useless columns. A column is useful if its variable has a non-zero solution or if the associated dual constraint is tight.

Precomputing Columns

If we consider small projections and combine their equivalent labels, the number of different subproblems is limited. There are only 33 equivalence classes with two alive states; only 744 with three; and 44076 with four. As binary variables are quite common and projections to one or two variables often carry a lot of the information, these subproblems are especially important.

The double description method (Motzkin et al. 1953) can be used to compute a *skeleton* of the subproblem (the set of rays used in the Minkowski sum). We found that small abstractions have small skeletons with at most two rays when the projection has two states and up to five rays when it has three. This is encouraging for two reasons: *(i)* subproblems cannot generate many columns, so the Dantzig-Wolfe decomposition is likely to converge quickly, and *(ii)* it is feasible to precompute and store skeletons for all small abstractions. At runtime, their pricing problems could then be replaced by a method that simply checks if any ray in the skeleton has a negative objective value. If so, that ray is returned, otherwise all constraints in the pricing problem are satisfied. Checking a small number of precomputed rays is likely much faster than running an LP solver for the subproblem.

Alternatively, the double description method could be used at runtime for all abstractions that are sufficiently small. This is usually not done because the skeletons are generally assumed to be much larger.

Avoiding Search Altogether

We identified two interesting algorithmic challenges that deserve additional attention in future work.

Pattern discovery problem Given an operator count y, find an abstraction that has no flow consistent with y.

Pricing problem Given an operator count y and an abstraction, find a cost function that shows that y is no flow.

The pattern discovery problem could also be phrased in terms of a multiset of operators if the RMP solves the operator-counting MIP instead of its LP relaxation.

With efficient algorithms for both of these problems, the Dantzig-Wolfe method could be used as a planner. Consider running the method on the set of *all* abstractions. However, instead of generating all abstractions, we start with a small set, generate columns from it using the pricing algorithm, and extend it with the pattern discovery algorithm once no new column can be generated anymore. As soon as the pattern discovery algorithm has no solution, we know that the current RMP solution is the cost of an optimal plan.

Towards the goal of finding an efficient algorithm for the pricing problem it is worth investigating its polyhedron more closely. Observe that the constraints imply that there cannot be negative cost cycles in the abstraction because otherwise $h^{\alpha_i}(I, cost_i) = -\infty$ and there is no smaller $h_i \in \mathbb{R}$. Cost functions that do not induce negative cost cycles can always be scaled by a non-negative factor which scales the heuristic value under that cost function in the same way. Instead of considering $h_i \in \mathbb{R}$ we could limit the choices to $h_i \in \{-1, 0, 1\}$. With this condition, we could replace $P_i(y)$ by three problems $P_i(y, h_i)$ for $h_i \in \{-1, 0, 1\}$ that fix the value of h_i (ignoring cases where $P_i(y, h_i)$ is infeasible). Their objective then simplifies to minimizing $cost_i(y)$. The question whether $P_i(y)$ can be solved without an LP solver remains open but one approach could be to solve $P_i(y, h_i)$, i.e., find a saturated cost function $cost_i$ that minimizes $cost_i(y)$ and satisfies $h^{\alpha_i}(I, cost_i) = h_i$.

Conclusions

We applied Dantzig-Wolfe decomposition to the optimal cost partitioning of abstraction heuristics. The relaxed master problem can be seen as either finding a cost partition with a linear combination of some candidate cost functions or as an operator-counting LP over some post-hoc optimization constraints using alternative costs. The pricing problems are parameterized with an operator-count y that optimizes the current RMP and compute a new column iff their abstraction does not have a network flow consistent with y.

Empirically, we have seen that this method computes the optimal cost partitioning faster than previous methods and also gives good approximations early on. We have investigated how combining equivalent labels and restricting attention to interesting patterns improves cost partitioning methods in general and ours in particular. We have also explored several relaxations of the heuristic that make it faster to evaluate while not losing too much accuracy. Extensions used in other state-of-the-art heuristics could be added here as well, e.g., internal resource limits, diversification, or adding/removing cost partitions during the search.

Acknowledgments

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

References

Benders, J. F. 1962. Partitioning Procedures for Solving Mixed-variables Programming Problems. *Numerische Mathematik* 4(1): 238–252.

Corrêa, A. B.; and Pommerening, F. 2019. An Empirical Study of Perfect Potential Heuristics. In Lipovetzky, N.; Onaindia, E.; and Smith, D. E., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 114–118. AAAI Press.

Culberson, J. C.; and Schaeffer, J. 1998. Pattern Databases. *Computational Intelligence* 14(3): 318–334.

Dantzig, G. B.; and Wolfe, P. 1960. Decomposition principle for linear programs. *Operations Research* 8: 101–111.

Desrosiers, J.; and Lübbecke, M. E. 2005. A Primer in Column Generation. In Desaulniers, G.; Desrosiers, J.; and Solomon, M. M., eds., *Column Generation*, 1–32.

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

Ford, L. R.; and Fulkerson, D. R. 1958. A Suggested Computation for Maximal Multi-Commodity Network Flows. *Management Science* 5(1): 97–101.

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.

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

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

Motzkin, T. S.; Raiffa, H.; Thompson, G. L.; and Thrall, R. M. 1953. The Double Description Method. *Contributions to the Theory of Games (AM-28)* II: 51–74.

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

Pommerening, F.; Keller, T.; Halasi, V.; Seipp, J.; Sievers, S.; and Helmert, M. 2021a. Code, benchmarks and experiment data for the ICAPS 2021 paper "Dantzig-Wolfe Decomposition for Cost Partitioning". https://doi.org/10.5281/zenodo.4600642.

Pommerening, F.; Keller, T.; Halasi, V.; Seipp, J.; Sievers, S.; and Helmert, M. 2021b. Dantzig-Wolfe Decomposition for Cost Partitioning: Technical Report. Technical Report CS-2021-001, University of Basel, Department of Mathematics and Computer Science.

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

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

Pommerening, F.; Röger, G.; Helmert, M.; Cambazard, H.; Rousseau, L.-M.; and Salvagnin, D. 2019. Lagrangian Decomposition for Optimal Cost Partitioning. In Lipovetzky, N.; Onaindia, E.; and Smith, D. E., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 338–347. AAAI Press.

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

Seipp, J. 2019. Pattern Selection for Optimal Classical Planning with Saturated Cost Partitioning. In Kraus, S., ed., *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 5621–5627. IJCAI.

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

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

Sievers, S. 2018. Merge-and-Shrink Heuristics for Classical Planning: Efficient Implementation and Partial Abstractions. In Bulitko, V.; and Storandt, S., eds., *Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS 2018)*, 90–98. AAAI Press.

Sievers, S.; Pommerening, F.; Keller, T.; and Helmert, M. 2020. Cost-Partitioned Merge-and-Shrink Heuristics for Optimal Classical Planning. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*, 4152–4160. IJCAI.

Thüring, A. 2019. *Evaluation Of Post-Hoc Optimization Constraints Under Altered Cost Functions*. Master's thesis, University of Basel.