Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

Lagrangian Decomposition for Optimal Cost Partitioning

Florian Pommerening, Gabriele Röger, Malte Helmert
University of Basel
Basel, Switzerland
{florian.pommerening,gabriele.roeger,malte.helmert}@unibas.ch

Hadrien Cambazard
Univ. Grenoble Alpes, Grenoble INP, G-SCOP
38000 Grenoble, France
hadrien.cambazard@grenoble-inp.fr

Louis-Martin Rousseau
Polytechnique Montreal
Montreal, Canada
louis-martin.rousseau@polymtl.ca

Domenico Salvagnin
University of Padua
Padua, Italy
domenico.salvagnin@unipd.it

Abstract

Optimal cost partitioning of classical planning heuristics has been shown to lead to excellent heuristic values but is often prohibitively expensive to compute. Lagrangian decomposition and Lagrangian relaxation are classical tools in mathematical programming that apply to optimization problems with a special block structure. We analyze the application of Lagrangian decomposition to cost partitioning in the context of operator-counting heuristics and interpret Lagrangian multipliers as cost functions for the combined heuristics. This allows us to view the computation of an optimal cost partitioning as an iterative process that can be seeded with any cost partitioning and improves over time. We derive an anytime algorithm to compute an optimal non-negative cost partitioning of abstraction heuristics without involving an LP solver. In each iteration, the computation reduces to independent shortest path problems in all abstractions. Finally, we discuss the extension to general cost functions.

Introduction

In optimal planning with heuristic search, high-quality admissible heuristics are required to solve hard tasks. When individual heuristics cannot achieve a sufficiently high quality, multiple heuristics can be combined with cost partitioning (Katz and Domshlak 2007; Yang et al. 2008; Katz and Domshlak 2010). It allows us to additively combine several admissible heuristics by splitting up the original cost function and evaluating the heuristics under reduced costs.

Katz and Domshlak (2010) show that computing the best way of partitioning the cost among explicit-state abstraction heuristics is polynomial in the size of the abstract state spaces. Allowing general cost functions can increase the quality of the combined heuristic even further (Pommerening et al. 2015) while the computation remains polynomial. Pommerening (2017) points out that abstractions based on small projections using only up to three variables already produce near-perfect heuristic values in a lot of cases. The high quality of these heuristic values is unfortunately offset by a high computation time that has prevented their use in practice so far. While computing the optimal cost partitioning is polynomial in these cases, it requires solving a large linear program (LP) that scales with the number of transitions in the abstract state spaces.

Operator-counting heuristics (Pommerening et al. 2014) offer an alternative way to additively combine admissible heuristics. Abstraction heuristics can be expressed as operator-counting heuristics and their combination in one LP computes their optimal cost partitioning. The operator-counting LP is dual to the LP computing the optimal cost partitioning but it can be expressed in a more compact form (Pommerening, Helmert, and Bonet 2017). However, even in compact form the LPs are too large to make their computation worthwhile in practice for a large number of heuristics.

We apply a classical tool from mathematical programming called Lagrangian decomposition to operator-counting heuristics. This technique exploits a block structure in an LP that can easily be introduced in operator-counting LPs. The resulting LP can be seen as maximizing the sum of smaller independent subproblems over a set of parameters, called Lagrangian multipliers. We show that in the context of operator-counting, the Lagrangian multipliers correspond to cost functions for each subproblem and the subproblems correspond to the individual heuristics that are combined in the operator-counting framework. This is a way of proving that the combination of operator-counting heuristics computes their optimal cost partitioning. Additionally, this view of cost partitioning gives us access to other methods of solving the problem rather than solving the monolithic LP.

We consider subgradient optimization as a way to optimize the cost partitioning. This is an iterative process that evaluates all subproblems on the current cost partitioning and computes an update to their cost functions based on a subgradient. We present an algorithm based on it for the case of non-negative cost partitioning of abstraction heuristics. In this setting, the subgradient has a clear interpretation and can be computed by solving shortest path problems in the abstractions under different cost functions. Keeping track of the best cost partitioning seen so far turns this into an anytime algorithm that can be used to improve any suboptimal cost partitioning.

We also discuss how this algorithm could be extended to general cost partitioning that is not restricted to non-negative costs. In this general setting, higher heuristic values can be achieved but computing a subgradient is more complicated.

Lagrangian Relaxation and Decomposition

Lagrangian relaxation (Geoffrion 1974) is a classical technique in mathematical programming that allows structured relaxations to be constructed for optimization problems. Consider the following mathematical program P:

$$\begin{aligned} \min c^\top x \quad & \text{subject to} \\ Ax &\geq b \qquad (1) \\ x &\in \mathbb{X} \qquad (2) \end{aligned}$$

where we have split the set of constraints into two sets: the so-called *complicating* constraints (1) and some other *easy* constraints (2) that define the feasible region $x \in \mathbb{X}$. Lagrangian relaxation builds on the assumption that optimizing an arbitrary linear function over the set $\mathbb{X}$ is (significantly) easier than optimizing it over the feasible set of the original problem P. The Lagrangian approach relaxes the complicating constraints (1) into the objective function, with some non-negative multipliers $\lambda \in \mathbb{R}^m_{\geq 0}$ (where m is the number of complicating constraints), called *Lagrangian multipliers*. This yields the Lagrangian function:

$$\mathcal{L}(x, \lambda) = c^\top x + \lambda^\top (b - Ax)$$

For any choice of $\lambda \in \mathbb{R}^m_{\geq 0}$, the Lagrangian subproblem

$$\phi(\lambda) = \min \mathcal{L}(x, \lambda) \quad \text{subject to} \quad x \in \mathbb{X}$$

is a relaxation of P, and its optimal value $\phi(\lambda)$ gives a lower bound on the optimal value of P. In general, different values of λ give different lower bounds, so the task becomes to find the λ yielding the best possible lower bound, i.e., solving the *Lagrangian dual*:

$$\phi^* = \max \phi(\lambda) \quad \text{subject to} \quad \lambda \in \mathbb{R}^m_{\geq 0}$$

The Lagrangian dual function $\phi(\lambda)$ is a continuous concave function, that can be optimized by any nondifferentiable optimization algorithm, like the subgradient method which we describe in the next section.

Note that Lagrangian relaxation can take advantage of special block structures in P. For example, if the optimization problem decomposes into independent blocks after relaxing the complicating constraints, those blocks can be solved independently for any choice of λ. Interestingly, a modeling manipulation, called *Lagrangian decomposition* (Guignard and Kim 1987), allows the method to be applied also to the case in which the original problem has no obvious complicating constraints, but rather a set of linking variables. Let us consider a linear program P' of the form:

$$\begin{aligned} \min c^\top x \quad & \text{subject to} \\ & A_1 x + B_1 y_1 \geq b_1 \\ & \cdots \\ & A_k x + B_k y_k \geq b_k \\ & x, y_1, \ldots, y_k \geq 0 \end{aligned}$$

Note that the model would decompose into k independent blocks if we could get rid of the shared variables x. The basic idea of Lagrangian decomposition is to create additional copies (x_i) of the shared variables x, one for each block, and then impose that all those copies take the same value in all blocks. The reformulation reads:

$$\begin{aligned} \min c^\top x \quad & \text{subject to} \\ & A_1 x_1 + B_1 y_1 \geq b_1 \\ & \cdots \\ & A_k x_k + B_k y_k \geq b_k \\ & x = x_1 \\ & \cdots \\ & x = x_k \\ & x, x_1, \ldots, x_k, y_1, \ldots, y_k \geq 0 \end{aligned}$$

Clearly, this is equivalent to the original problem P'. Now, we can directly apply the Lagrangian relaxation technique to the reformulated problem, by dualizing the equality constraints $x = x_i$ with dual multipliers $\lambda_i \in \mathbb{R}^m$. The Lagrangian function reads:

$$\begin{aligned} \mathcal{L}(x, x_1, \ldots, x_k, \lambda_1, \ldots, \lambda_k) = \\ (c - \sum_{i=1}^k \lambda_i)^\top x + \sum_{i=1}^k \lambda_i^\top x_i \end{aligned}$$

and, for a given λ, it can be minimized by solving $k + 1$ independent subproblems. The first has the special structure:

$$\phi_0(\lambda) = \min (c - \sum_{i=1}^k \lambda_i)^\top x \quad \text{subject to} \quad x \geq 0$$

and it is always either optimal with objective value zero or unbounded. In other words, the first block defines the implicit set of constraints on the multipliers λ:

$$\sum_{i=1}^k \lambda_i \leq c$$

The other k subproblems all have the same structure:

$$\begin{aligned} \phi_i(\lambda_i) = \min \lambda_i^\top x_i \quad & \text{subject to} \\ A_i x_i + B_i y_i &\geq b_i \\ x_i, y_i &\geq 0 \end{aligned}$$

The value of our original problem P' then is

$$\phi^* = \max \sum_{i=1}^k \phi_i(\lambda_i) \quad \text{subject to} \quad \sum_{i=1}^k \lambda_i \leq c.$$

Subgradient methods

The subgradient method (Shor 1985) is an extension to the gradient method for differentiable optimization to the nondifferentiable case. Given a continuous nondifferentiable concave function $\phi(\lambda)$ and a convex set Ω, the subgradient method can, under mild conditions (Anstreicher and Wolsey 2009), solve the problem

$$\max \phi(\lambda) \quad \text{subject to} \quad \lambda \in \Omega.$$

The function to be maximized ϕ need only be known through a black box oracle that, given a point $\lambda \in \Omega$, returns the function value $\phi(\lambda)$ and a subgradient $g \in \partial\phi(\lambda)$, where $\partial\phi(\lambda)$ denotes the subdifferential of ϕ at λ. Then, the subgradient algorithm is a simple iterative procedure, that, starting from an initial point $\lambda^{(0)} \in \Omega$, uses the simple update formula:

$$\lambda^{(t+1)} = P_\Omega\left(\lambda^{(t)} + \eta(t)g^{(t)}\right)$$

where $\eta(t) > 0$ is the step length at iteration t, $g^{(t)}$ is a subgradient of ϕ at $\lambda^{(t)}$, and $P_\Omega(\cdot)$ denotes the projection on Ω. Under mild assumptions on the step length sequence η, the method converges to an optimal solution, e.g. with $\lim_{t\to\infty}\eta(t) = 0$, $\sum_t \eta(t) = \infty$, $\sum_t \eta(t)^2 < \infty$, and bounded $\|g\|$ (Anstreicher and Wolsey 2009). Clearly, the method is a viable choice if it is relatively inexpensive to (a) compute a subgradient g at an arbitrary point $\lambda \in \Omega$ and (b) project a point onto Ω.

Consider the case where $\phi(\lambda)$ is the Lagrangian subproblem we got by Lagrangian decomposition in the previous section. The subproblems ϕ_i are independent, so concatenating subgradients of ϕ_i gives a subgradient of ϕ. A subgradient g of ϕ_i at λ_0 is a vector that satisfies $\phi_i(\lambda_0) - \phi_i(\lambda) \geq g(\lambda_0 - \lambda)$ for all λ.[1] Let x_i^* be an optimal solution of subproblem $\phi_i(\lambda_0)$. Then $\phi_i(\lambda_0) = x_i^*\lambda_0$. Since x_i^* is a (not necessarily optimal) solution of $\phi_i(\lambda)$, we have $\phi_i(\lambda) \leq x_i^*\lambda$. Thus $\phi_i(\lambda_0) - \phi_i(\lambda) \geq x_i^*\lambda_0 - x_i^*\lambda = x_i^*(\lambda_0 - \lambda)$ and x_i^* is a subgradient of ϕ_i at λ_0. In other words, the optimal solutions of the subproblems ϕ_i are subgradients.

Subgradient methods are not very practical for obtaining high accuracy: they require $\Theta(1/\epsilon^2)$ iterations to get to an absolute error ϵ (Frangioni, Gendron, and Gorgone 2017). On the one hand this is the best possible complexity for optimizing a function only through a black box, but on the other hand it cannot compete with other more sophisticated algorithms, like Bundle methods (Hiriart-Urruty and Lemaréchal 1993) or center methods (du Merle, Goffin, and Vial 1998). Still, their convergence rate does not depend on the number of variables, so they are a promising option to get approximate bounds for large scale optimization problems in short computing times.

Interpretation in the Operator-Counting Framework

Operator-counting heuristics (Pommerening et al. 2014) are a class of heuristics that can be expressed in a certain mathematical form. They are defined as the objective value of a linear program which uses variables count_o that count how often an operator o is used (so-called *operator-counting variables*) and some auxiliary variables y_i that are not shared between constraints and can have different interpretations in different constraints:

$$\begin{aligned} \min \mathit{cost}^\top \mathit{count} \quad & \text{subject to} \\ & A_1\mathit{count} + B_1 y_1 \geq b_1 \\ & \cdots \\ & A_k\mathit{count} + B_k y_k \geq b_k \\ & \mathit{count}, y_1, \ldots, y_k \geq 0 \end{aligned}$$

To represent an admissible heuristic, every constraint $A_i\mathit{count} + B_i y_i \geq b_i$ must have a solution for every plan π, where count_o matches the number of times operator o is used in π. If a constraint has this property, it represents a necessary property of all plans and is called an *operator-counting constraint*. The LP minimizes the cost of a selection of operators that satisfy all specified necessary properties of plans. While this selection is not necessarily a plan, its cost is a lower bound since all plans (including optimal ones) must satisfy the constraints.

Different heuristics can be expressed as operator-counting constraints (Pommerening et al. 2014). For example, the constraint $\sum_{o\in L} \mathit{count}_o \geq 1$ expresses that a set of operators L is a disjunctive action landmark. An abstraction heuristic is defined as the cost of a shortest path in an abstraction of the planning task. After removing dead states, it can be expressed with constraints that balance the incoming and outgoing flow of each abstract state s using auxiliary variables y_t for each abstract transition:

$$\sum_{t\in \mathit{in}(s)} y_t - \sum_{t\in \mathit{out}(s)} y_t = \Delta_s \qquad \text{for all abstract states } s$$

$$\sum_{t \text{ labeled with } o} y_t = \mathit{count}_o \qquad \text{for all operators } o$$

where $\mathit{in}(s)$ and $\mathit{out}(s)$ are the sets of incoming and outgoing transitions of s and Δ_s is -1, 1, or 0 depending on whether s is the abstraction of the current state, a goal state, both, or neither. Note that all of these equations for one abstraction together form a single operator-counting constraint that uses additional variables y_t for each transition.

While individual heuristics can be expressed with individual operator-counting constraints, we can use constraints from multiple heuristics in the same LP. This forces solutions to respect properties of all heuristics simultaneously, i.e., with the same number of of times an operator is used in each heuristic. For example, using shortest path constraints from multiple abstractions forces the solution to be a selection of operators that can be used as a path in all abstractions. Such a solution has at least the value of the maximum over all heuristics but it can have a higher value. In fact, Pommerening et al. (2015) prove that this combination always computes an optimal cost partitioning over the participating heuristics $h_1, \ldots, h_k$, where the heuristic h_i is defined as the solution of the following LP:

$$\begin{aligned} \min \mathit{cost}^\top \mathit{count} \quad & \text{subject to} \\ & A_i\mathit{count} + B_i y_i \geq b_i \\ & \mathit{count}, y_i \geq 0 \end{aligned}$$

In general, a cost partitioning (Katz and Domshlak 2007; Yang et al. 2008; Katz and Domshlak 2010) is a distribution

[1] Technically, this would be a supergradient, since we are maximizing the concave function ϕ instead of minimizing a convex function. The term subgradient is still commonly used in this case, just like "hill-climbing" is used for minimization problems.

of the operator costs among k heuristic computations such that $\sum_{i=1}^{k} \mathit{cost}_i \leq \mathit{cost}$. If the sub-heuristics are admissible, the sum of their estimates (under the modified cost) is an admissible estimate under the original cost function. We distinguish *general cost partitioning* where cost_i can be negative from *non-negative cost partitioning*. A cost partitioning is optimal if it maximizes the sum of estimates.

We now apply Lagrangian decomposition to operator-counting heuristics. First, we have to duplicate variables to induce the block structure that Lagrangian relaxation requires. In our case, we only duplicate the operator-counting variables since the additional variables y_i are not shared among operator-counting constraints:

$$\begin{aligned} \min \mathit{cost}^\top \mathit{count} \quad & \text{subject to} \\ & A_1 \mathit{count}_1 + B_1 y_1 \geq b_1 \\ & \cdots \\ & A_k \mathit{count}_k + B_k y_k \geq b_k \\ & \mathit{count} = \mathit{count}_1 \\ & \cdots \\ & \mathit{count} = \mathit{count}_k \\ \mathit{count}, \mathit{count}_1, \ldots, \mathit{count}_k, y_1, \ldots, y_k \geq 0 & \end{aligned}$$

We relax each constraint $\mathit{count} = \mathit{count}_i$ with an unrestricted multiplier we call cost_i. The Lagrangian function is:

$$\begin{aligned} & \mathcal{L}(\mathit{count}, \mathit{count}_1, \ldots, \mathit{count}_k, \mathit{cost}_1, \ldots, \mathit{cost}_k) \\ & = (\mathit{cost} - \sum_{i=1}^{k} \mathit{cost}_i)^\top \mathit{count} + \sum_{i=1}^{k} \mathit{cost}_i^\top \mathit{count}_i. \end{aligned}$$

Again, for fixed values of the multipliers cost_i, the Lagrangian function can be evaluated by solving $k+1$ independent subproblems. As described before, the first subproblem has the special structure

$$\min (\mathit{cost} - \sum_{i=1}^{k} \mathit{cost}_i)^\top \mathit{count} \quad \text{subject to} \quad \mathit{count} \geq 0$$

which implicitly defines a constraint on the multipliers that matches the cost partitioning constraint $\sum_{i=1}^{k} \mathit{cost}_i \leq \mathit{cost}$. The remaining k subproblems all have the same structure:

$$\begin{aligned} \min \mathit{cost}_i^\top \mathit{count}_i \quad & \text{subject to} \\ & A_i \mathit{count}_i + B_i y_i \geq b_i \\ & \mathit{count}_i, y_i \geq 0. \end{aligned}$$

This is exactly the definition of the heuristic h_i that uses the single operator-counting constraint $A_i \mathit{count} + B_i y_i \geq b_i$.

Putting all parts together, we see that applying Lagrangian decomposition to the operator-counting LP yields the Lagrangian dual function $\phi(\mathit{cost}_1, \ldots, \mathit{cost}_k)$ that has a value of $\sum_{i=1}^{k} h_i(\mathit{cost}_i)$ if the cost functions cost_i form a general cost partitioning and becomes unbounded if they do not.

Optimizing the values of the multipliers, i.e., computing the optimal value of the Lagrangian dual function $\phi^* = \max \phi(\mathit{cost}_1, \ldots, \mathit{cost}_k)$, computes an optimal cost partitioning of the heuristics h_i. The Lagrangian multipliers directly correspond to cost functions for each subproblem. This is not too surprising, as Pommerening et al. (2015) already used a similar argument to prove the connection of operator counting and cost partitioning. However, they did not draw the connection to Lagrangian decomposition which allows us to use subgradient optimization to compute optimal cost partitionings. We specialize the subgradient optimization algorithm for this use case (using planning notation):

1. Let $\mathit{cost}_i^{(1)}$ for $1 \leq i \leq k$ be a cost partitioning, i.e., cost functions that satisfy $\sum_{i=1}^{k} \mathit{cost}_i^{(1)} \leq \mathit{cost}$. Then repeat the following steps for $t = 1, 2, \ldots$.
2. Compute $h_i(\mathit{cost}_i^{(t)})$ for all $1 \leq i \leq k$. Every optimizing solution $\mathit{count}_i^{*(t)}$ is a subgradient of h_i at $\mathit{cost}_i^{(t)}$.
3. Follow the subgradient for a step with length $\eta(t)$, i.e., set $c_i^{(t+1)} = \mathit{cost}_i^{(t)} + \eta(t) \mathit{count}_i^{*(t)}$ for $1 \leq i \leq k$.
4. Project the resulting cost functions to the space of cost partitionings, i.e., set $\mathit{cost}^{(t+1)} = P_\Omega(c_1^{(t+1)}, \ldots, c_k^{(t+1)})$, where $\Omega = \{\langle c_1, \ldots, c_k \rangle \in \mathbb{R}^{k|O|} \mid \sum_{i=1}^{k} c_i \leq \mathit{cost}\}$. We will discuss the implementation of P_Ω later.

Intuitively, this means that we start from any cost partitioning and then – step by step – adjust the costs in a certain direction until we end up in an optimal cost partitioning. The direction in which the costs are adjusted for each operator o depends on how important it is for an optimal solution for each of the component heuristics under the current cost partitioning. For simplicity assume there are just two heuristics: if the operator is used the same number of times in both heuristics, its cost needs no adjustment; if it is used more in h_1 then it makes sense to give it a higher cost in h_1; otherwise it should get a higher cost in h_2.

At any iteration, the process can be stopped and the best cost partitioning we have observed so far yields a lower bound to the optimal cost partitioning. Since we can start from any cost partitioning, this method can also be seen as a way to improve suboptimal cost partitioning methods.

Cost Partitioning of Abstraction Heuristics

Let us consider a special case of the general algorithm above where the heuristics h_i are abstractions. In that case, the heuristic value under any cost function is the cost of an optimal plan in the abstract state space and an optimizing solution of the operator-counting variables is the number of times each operator occurs in this plan. This leads to an algorithm for computing the optimal cost partitioning of abstraction heuristics that does not involve an LP solver:

1. Let $\mathit{cost}_i^{(1)}$ for $1 \leq i \leq k$ be a cost partitioning and repeat the following steps for $t = 1, 2, \ldots$
2. Compute an optimal plan $\pi_i^{(t)}$ under cost function $\mathit{cost}_i^{(t)}$ for each abstraction α_i and let $\mathit{occurrences}(o, \pi_i^{(t)})$ be the number of occurrences of o in $\pi_i^{(t)}$. The vector of these numbers for all operators is a subgradient of h^{α_i} at $\mathit{cost}_i^{(t)}$.
3. Follow the subgradient for a step with length $\eta(t)$, i.e., set $c_i^{(t+1)}(o) = \mathit{cost}_i^{(t)}(o) + \eta(t) \mathit{occurrences}(o, \pi_i^{(t)})$.
4. Project the cost functions to cost partitionings.

Projecting to Cost Partitionings

The above methods rely on a projection to the space Ω of general cost partitionings. Such a projection is easy to define in principle. We start from cost functions $cost'_1, \ldots, cost'_k$ that do not form a cost partitioning. We are looking for new cost functions $cost_1, \ldots, cost_k$ that respect the cost partitioning constraint $\sum_{i=1}^{k} cost_i \leq cost$ and minimize the Euclidean distance to $(cost'_1, \ldots, cost'_k)$. We can consider each operator o individually since the choices for each of them are independent. For o to satisfy the cost partitioning constraint, the sum of its cost has to be reduced by $r_o = \max\{0, \sum_{i=1}^{k} cost'_i(o) - cost(o)\}$. If we use $\Delta_{i,o}$ as the value for which $cost_i(o) = cost'_i(o) - \Delta_{i,o}$, then satisfying the cost partitioning constraint can be rephrased as distributing r_o among all operators: $\sum_{i=1}^{k} \Delta_{i,o} \geq r_o$ The problem is to find $\Delta_{i,o}$ that minimize the Euclidean distance, i.e., minimizing $\sum_{i=1}^{k} (cost'_i(o) - cost_i(o))^2 = \sum_{i=1}^{k} \Delta_{i,o}^2$. The sum of squares is minimized if the elements all have the same size, so setting $\Delta_{i,o} = r_o/k$ is the desired projection.

Projecting cost functions like this will quickly lead to negative operator costs in some abstractions. This can be problematic if subproblems can become unbounded under negative cost functions. For example, if an abstraction contains a cycle of transitions with negative total cost on a path from the abstract initial state to an abstract goal state, there is no shortest path in the abstract task. We can construct arbitrarily cheap plans by repeating the cycle often enough. The heuristic value is defined as $-\infty$ in this case, thus the heuristic remains admissible. However, we can no longer extract a subgradient from the optimal solution.

The reason why this can lead to a problem is that each subproblem implicitly defines a constraint $h_i(cost_i) > -\infty$ on the Lagrangian multipliers $cost_i$. For example, in abstraction heuristics, the constraint prohibits all cost functions that induce a cycle with negative total cost in the abstraction. Ideally, we would like to project to the space $\Omega' = \{\langle cost_1, \ldots, cost_k \rangle \in \Omega \mid h_i(cost_i) > -\infty \text{ for } 1 \leq i \leq k\}$. For abstraction heuristics, this would amount to distributing the cost changes in a way that not only reduces the total cost of each operator by r_o but also reduces the cost of each cycle Z by at most $a_Z = \sum_{o \in Z} cost'_i(o)$. The projection to Ω' is a solution to the following problem:

$$\min \sum_{i=1}^{k} \Delta_{i,o}^2 \quad \text{subject to}$$

$$\sum_{i=1}^{k} \Delta_{i,o} \geq r_o \qquad \text{for all operators } o$$

$$\sum_{o \in Z} \Delta_{i,o} \leq a_Z \qquad \text{for all } 1 \leq i \leq k \text{ and all cycles } Z \text{ in abstraction } i$$

This projection is a quadratic optimization problem with an exponential number of constraints. In this form, we cannot solve it efficiently. We leave finding a general projection method for future work and discuss some possible approaches to this problem at the end of the paper.

For now, we focus on the traditional *non-negative* cost partitioning, where costs are restricted to non-negative numbers. Here, projection is more straight-forward. The projection has to satisfy the additional constraint $\Delta_{i,o} \leq cost'_i(o)$ for all subproblems i and operators o. We still have to distribute cost differences as evenly as possible but now certain operators might not be able to lower their cost by r_o/k.

We compute the projection for o by repeating the following steps until $r_o = 0$:

1. Compute $\Delta = \frac{r_o}{|\{i | cost'_i(o) > 0\}|}$
2. Reduce $cost'_i(o)$ by $\min\{cost'_i(o), \Delta\}$ for every abstraction i
3. Recompute r_0 with the updated $cost'$ values

The first step tries to distribute the necessary cost as evenly as possible among all operators that still have positive costs. The second step ensures that no cost is reduced below 0. If $r_o = 0$ for all operators o, $cost'$ is a non-negative cost partitioning and we can use it for the desired projection.

Theorem 1. *The cost partitioning computed by the procedure above minimizes the Euclidean distance to the set of all non-negative cost partitionings.*

Proof sketch. Let A_0 be the set of indices i for which $cost'_i(o) = 0$ after the procedure terminated and $A_{>0} = \{1, \ldots, k\} \setminus A_0$ be the set where the operator's cost remains positive. Let c_i denote the value of $cost'_i(o)$ before the procedure started. Let Δ_i be the total amount by which $cost'_i(o)$ has been reduced by the procedure.

For $i, j \in A_{>0}$, $\Delta_i = \Delta_j$ (*) because every iteration reduced the cost by Δ, as computed in step 1. For $i \in A_{>0}$ and $j \in A_0$, $\Delta_i \geq \Delta_j = c_j$ (**). We need to show that these values minimize $S := \sum_{i=1}^{k} \Delta_i^2$ under the additional constraint that costs stay non-negative, i.e., $\Delta_i \leq c_i$ for all i. Due to this constraint, for $i \in A_0$ the value for Δ_i cannot be increased. If it is decreased for some $i \in A_0$ then the value for one or more $j \in A_{>0}$ must be increased. Due to (**), this would increase S. Only modifying elements of $A_{>0}$ also would increase S due to (*). □

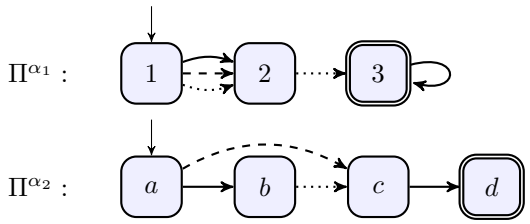
Example

Figure 1 shows an example of the subgradient algorithm that optimizes a cost partitioning among two abstractions α_1 and α_2 in five steps. The top of the figure shows the abstract transition systems of the two abstractions. There are three operators $\longrightarrow$, $\text{- -}\!\!\rightarrow$, and $\cdots\!\!\rightarrow$ that all have a cost of 1. The cost partitioning is initialized to the uniform cost partitioning which assigns each operator a cost of 0.5 in each abstraction because all operators are relevant to both abstractions.

We use $\eta(t) = 1/t$ for the step length at iteration t, which guarantees convergence to an optimal solution.

In the first iteration, both abstractions have a shortest plan of cost 1. In α_1 there are three possible shortest plans and we assume the algorithm discovered $\langle \longrightarrow, \cdots\!\!\rightarrow \rangle$. In α_2 there is only one choice: $\langle \text{- -}\!\!\rightarrow, \longrightarrow \rangle$.

With step length $\eta(1) = 1$, the cost of $\longrightarrow$ is increased by 1 in both abstractions, while the cost of $\cdots\!\!\rightarrow$ is only increased in α_1 and that of $\text{- -}\!\!\rightarrow$ only in α_2. The resulting cost functions



t		α_1			α_2		
		$\longrightarrow$	$\dashrightarrow$	$\cdots\blacktriangleright$	$\longrightarrow$	$\dashrightarrow$	$\cdots\blacktriangleright$
1	*cost*	0.5	0.5	0.5	0.5	0.5	0.5
	$\pi/cost(\pi)$	$\langle\longrightarrow,\cdots\blacktriangleright\rangle/1$			$\langle\dashrightarrow,\longrightarrow\rangle/1$		
	count*	1	0	1	1	1	0
	cost$'$	1.5	0.5	1.5	1.5	1.5	0.5
2	*cost*	0.5	0	1	0.5	1	0
	$\pi/cost(\pi)$	$\langle\dashrightarrow,\cdots\blacktriangleright\rangle/1$			$\langle\longrightarrow,\cdots\blacktriangleright,\longrightarrow\rangle/1$		
	count*	0	1	1	2	0	1
	cost$'$	0.5	0.5	1.5	1.5	1	0.5
3	*cost*	0	0.25	1	1	0.75	0
	$\pi/cost(\pi)$	$\langle\longrightarrow,\cdots\blacktriangleright\rangle/1$			$\langle\dashrightarrow,\longrightarrow\rangle/1.75$		
	count*	1	0	1	1	1	0
	cost$'$	$0.\bar{3}$	0.25	$1.\bar{3}$	$1.\bar{3}$	$1.08\bar{3}$	0
4	*cost*	0	$0.08\bar{3}$	1	1	$0.91\bar{6}$	0
	$\pi/cost(\pi)$	$\langle\longrightarrow,\cdots\blacktriangleright\rangle/1$			$\langle\dashrightarrow,\longrightarrow\rangle/1.91\bar{6}$		
	count*	1	0	1	1	1	0
	cost$'$	0.25	$0.08\bar{3}$	1.25	1.25	$1.1\bar{6}$	0
5	*cost*	0	0	1	1	1	0
	$\pi/cost(\pi)$	$\langle\longrightarrow,\cdots\blacktriangleright,\longrightarrow\rangle/1$			$\langle\longrightarrow,\cdots\blacktriangleright,\longrightarrow\rangle/2$		

Figure 1: Example for five steps of the subgradient method with two abstractions α_1 and α_2. The transition systems of α_1 and α_2 are shown at the top and the evolution of the cost partitioning in the table below. All operators have the cost 1.

cost$'$ no longer satisfy the cost partitioning constraint and have to be projected back into the space of non-negative cost partitionings. For $\longrightarrow$ we reduce both costs by 1 and for the other operators, we reduce both costs by 0.5.

Under the new cost functions, different plans are now optimal in both abstractions ($\langle\dashrightarrow,\cdots\blacktriangleright\rangle$ and $\langle\longrightarrow,\cdots\blacktriangleright,\longrightarrow\rangle$). Both plans still have a cost of 1, so our modification of the cost functions did not increase the overall heuristic value yet. We increase the cost of $\dashrightarrow$ and $\cdots\blacktriangleright$ in α_1. They are both used once and our step length is now $\eta(2) = 0.5$ so we increase their cost by 0.5. In α_2 we increase the cost of $\cdots\blacktriangleright$ by 0.5 as well but increase the cost of $\longrightarrow$ by 1 because it is used twice.

After this step, the two shortest plans ($\langle\longrightarrow,\cdots\blacktriangleright\rangle$ and $\langle\dashrightarrow,\longrightarrow\rangle$) have a total cost of 2.75 showing that our cost partitioning improved. We update the costs of all used operators by $\eta(3) = 0.\bar{3}$ and project back to a cost partitioning.

In the fourth step the shortest plans are the same as before but the cost partitioning has improved again and has a total value of $2.91\bar{6}$. After updating the cost of all used operators by $\eta(4) = 0.25$, we project to a cost partitioning again. This time, the cost of $\dashrightarrow$ cannot be reduced by the same amount in α_1 and α_2, so in a first step its cost is reduced to 0 in α_1 and by 0.125 to $1.041\bar{6}$ in α_2. In a second step, we reduce it by the remaining surplus to a cost of 1 in α_2.

The final cost partitioning is optimal in this case and has a heuristic value of 3. There are different shortest plans under these cost functions but note that there are shortest plans that use each operator the same number of times. This corresponds to an optimal solution of the operator-counting LP where these numbers have to be the same because they are represented by the same variable. From the perspective of the subgradient method these plans are also interesting. They increase the cost of each operator by the same amount in each abstraction, which means the costs are projected back to the value they started from. This shows that **0** is a subgradient and we have reached an optimal value.

Experiments

We implemented the subgradient algorithm for abstraction heuristics in the Fast Downward planning system (Helmert 2006). We used projections as our choice of abstractions and explicitly construct their transition system in memory, pruning all abstract states that are not reachable or have no path to an abstract goal state. We implemented two methods to solve the shortest path problems in the abstractions. One uses Dijkstra's algorithm (Dijkstra 1959) on the abstract transition system, the other solves the operator-counting heuristic for the flow balance constraints of the abstract states. The number of abstract states in an abstraction never reached 10^5, so using Dijkstra's algorithm over, say, A* is no problem. Since the abstractions contain no dead states, the LP with flow balance constraints is equivalent to computing their shortest path (Pommerening, Helmert, and Bonet 2017). We compare this implementation to computing the optimal non-negative cost partitioning as one monolithic operator-counting LP using flow balance constraints of all abstractions. In all cases, we use CPLEX 12.8 to solve LPs. All LPs are only constructed once and kept in memory. From one iteration of the subgradient algorithm to the next, only the objective function is changed. The experiments use all planning tasks from optimal tracks of international planning competitions (IPC 1998–2018) that do not have conditional effects or axioms. We limit time to 300 s and memory to 2 GB for each instance.

We use the set of all projections to up to n variables (for pattern sizes $n \in \{2, 3\}$) and limit the patterns to those that are interesting (Pommerening, Röger, and Helmert 2013) in the context of non-negative cost partitioning. For a pattern size of 3 this is a large set of patterns that cannot be constructed for all planning tasks. However, we think this is a good choice for evaluating the subgradient algorithm because the large number of abstractions make the optimal cost partitioning hard to find.

The subgradient algorithm uses the step length $\eta(t) = 1/t$. We made this naïve and straight-forward choice to establish a baseline and to not dilute the main message of the paper. We leave the exploration of more advanced choices as future work. In a first experiment, we start from the uniform cost partitioning and look at how the best known heuristic

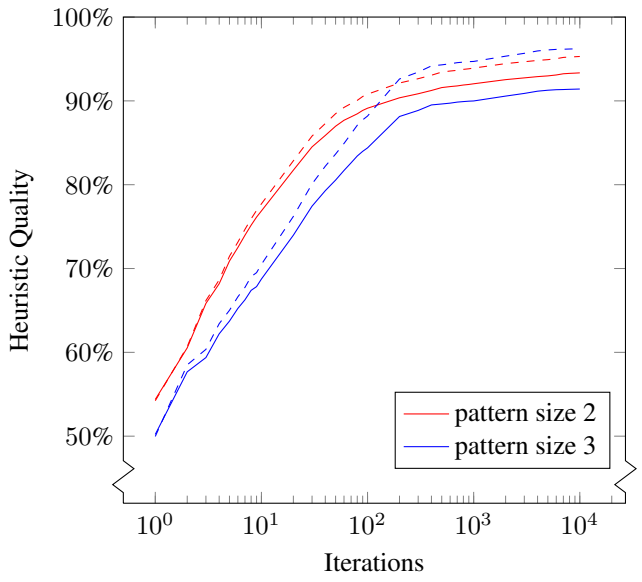


Figure 2: Heuristic quality of uniform cost partitioning improved using iterations of the subgradient method and measured as the geometric mean of the ratio $h^C(s)/h^{C^*}(s)$. Dashed lines ignore the domain ParcPrinter.

value changes with more iterations of the subgradient algorithm. We measure the quality of a cost partitioning C as the fraction $h^C(s)/h^{C^*}(s)$ where $h^C(s)$ is the heuristic value under C and $h^{C^*}(s)$ is the heuristic value under an optimal non-negative cost partitioning. We aggregate the quality values for the initial states of all instances where we can compute both h^C and h^{C^*} with the geometric mean. Figure 2 reports the aggregated heuristic quality after different numbers of iterations for the implementation that uses Dijkstra's algorithm for the subproblems. The quality increases quite steeply with the first few iterations. After 100 iterations, the quality already exceeds 89% for patterns of size 2 and 84% for patterns of size 3. After 1000 iterations it exceeds 92% and 89%. This is promising news showing that we get a large part of the benefit from the subgradient method already after a small number of iterations. Unfortunately, further iterations suffer from diminishing returns and after reaching roughly 90% accuracy, improvement slows down.

The heuristic error is worst in the domain ParcPrinter which has operator costs on the order of 10^6. These high costs show a problem of the subgradient method: the gradient only depends on the number of times an operator is used in a cheapest plan and otherwise ignores the operator cost. Optimal ParcPrinter plans are usually orders of magnitude shorter than their cost and do not repeat operators. With a step length of $\eta(t) = 1/t$ this means that the cost function is only modified by at most 1 in each step. If the initial uniform cost partitioning is far from optimal, it takes many iterations to move to a better one.

Uniform cost partitioning is easy to compute but not very accurate. We also initialized the subgradient method with other cost partitioning methods to see how the achieved quality compares to the state of the art. In particular, we used opportunistic uniform, greedy zero-one, and saturated cost partitioning (Seipp, Keller, and Helmert 2017a). All of these methods depend on an order of the abstractions and decide how much of the remaining costs to allocate to an abstraction one at a time. We tried all methods with a random order and one discovered by hill-climbing in the space of orders for 100 seconds (Seipp, Keller, and Helmert 2017b).

Figure 3 shows the quality of each cost partitioning method and how it can be improved with subgradient optimization. With greedy zero-one partitioning, bad orders can yield heuristic values of 0 which means the geometric mean over the qualities is 0. Since this would hide information of other tasks we use a heuristic value of 1 in such cases. We can see that the quality achieved when starting from uniform cost partitioning surpasses all tested cost partitioning methods in the first 200 iterations. and continues to improve afterwards. Additionally, starting from a better seed can lead to even better quality and even saturated cost partitioning with an improved order that already has a high quality can be further improved in a small number of steps.

The heuristic values were almost identical for the implementation that used an LP solver for its subproblems. Small differences can be caused by selecting different optimal plans in the abstractions. Over all, these differences seemed to cancel out and there was no systematic advantage of one method over the other. The main difference between the methods is their computation speed. Figure 4 shows that computing the shortest path with a specialized algorithm is between 10 and 200 times faster than using an LP solver.

We now compare the time it took to compute the optimal cost partitioning to the time it took to approximate it with the subgradient method. For the subgradient method, the time scales more or less linearly with the number of iterations, so the comparison relies on our desired level of accuracy. Figure 5 shows the time it took to compute 200 iterations which achieved accuracies of 90% and 88% for pattern sizes 2 and 3. For easy instances where the optimal cost partitioning can be computed in less than a second, the overhead of 200 subgradient iterations is too high and we observe some solving times increasing slightly. Limiting the number of iterations of the subgradient method is a very naïve stopping condition. Our algorithm will perform 200 iterations even if the uniform cost partitioning that we started from is already optimal. A more informed stopping condition would probably help to improve the runtime of such small instances. On larger instances, where the optimal cost partitioning is hard to find, we can see the benefit of the subgradient method.

We also ran experiments finding a cost partitioning in every state of the search. Due to the high number of projections and the difficulty of finding an optimal cost partitioning, only small tasks could be solved, where the subgradient method did not improve performance. Better results can probably be achieved by computing close-to-perfect cost partitionings for some sample states before starting the search and using those cost partitionings for all states. This approach is similar to computing high-dimensional potential heuristics but remains polynomial if we limit the number of subgradient iterations.

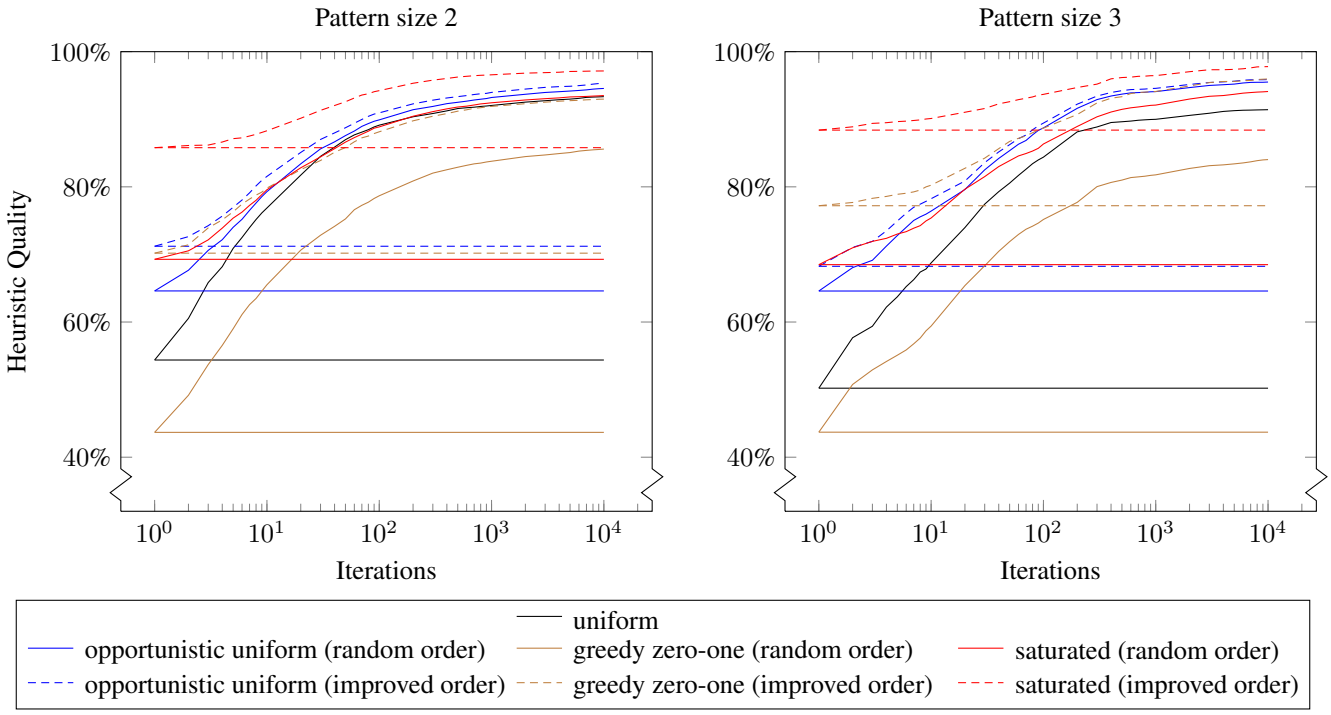


Figure 3: Heuristic quality of different suboptimal cost partitioning methods measured as the geometric mean of the ratio $h^C(s)/h^{C^*}(s)$. In each case, the horizontal line shows the quality of the cost partitioning method while the other line shows how the quality improves with subgradient optimization. We only consider instances where all methods finished 10000 iterations and the optimal cost partitioning could be computed.

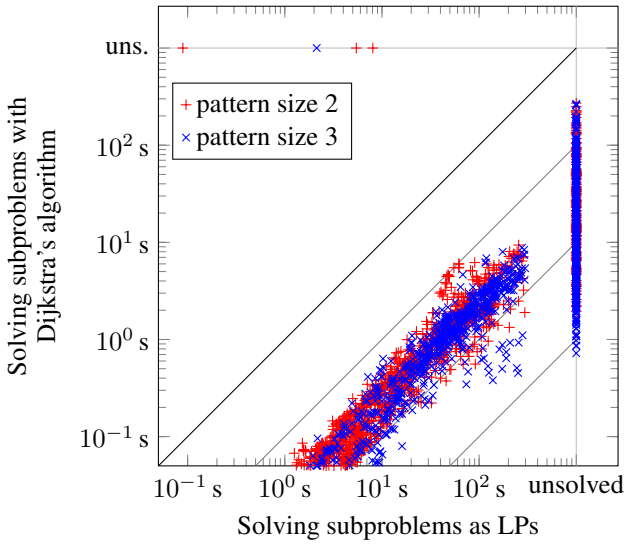


Figure 4: Time required to compute the heuristic value of the initial state using 200 iterations of the subgradient method. Grey lines mark factors of 10.

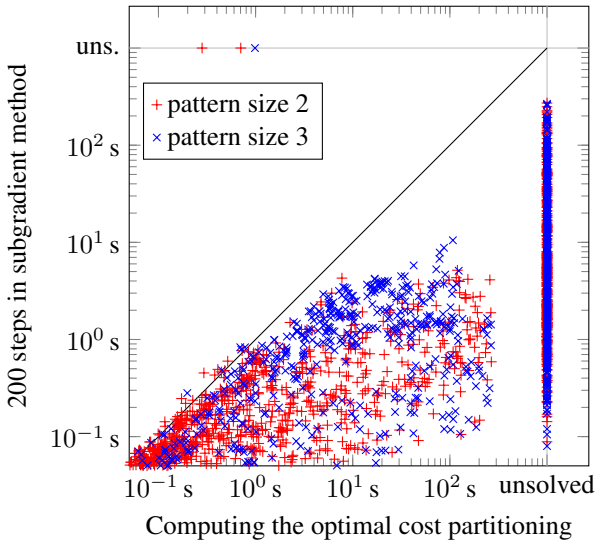


Figure 5: Time required to compute the heuristic value of the initial state using either 200 iterations of the subgradient method or solving the monolithic cost partitioning LP.

Future Work

We presented a general subgradient algorithm to improve suboptimal cost partitionings and an initial instantiation with straight-forward choices for the gradient, step length, and stopping condition. More elaborate choices are possible and there are many interesting directions to go from here.

We restricted our attention to non-negative cost partitioning. The extension to general costs relies on an efficient method for projecting arbitrary cost functions into the space of cost partitionings that do not induce negative cost cycles in the abstractions. We discussed the requirements for such a projection but have no efficient way of computing it yet.

Instead of viewing the absence of negative cost cycles in the abstractions as an implicit constraint on the Lagrangian multipliers, we can also modify the definition of the heuristic so it has a finite value for all cost functions. For example, limiting the number of times each transition can be used, will give a low finite heuristic value for cases where the cost function induces a negative cycle. If the limit is high enough, such cost partitionings are suboptimal and the subgradient method will move away from them. The shortest path will lead through the cycle making the gradient positive for the operators in the cycle. This increases their cost eventually getting rid off the negative cost cycle. However, with the extra constraints, the subproblems are no longer simple shortest path problems. Additionally, introducing limits makes the heuristic inadmissible. Dynamically raising the limits would be an option but further complicates the method.

The second large problem we saw in the subgradient method was that it does not consider the costs of operators causing it to suffer in domains like ParcPrinter. Normalizing the gradient and adjusting the step length function η according to the average operator costs would be interesting. However, it is not immediately clear how vastly different operator costs should be treated.

Finally, we did not explore choices for stopping conditions yet. We only reported the quality and time after fixed numbers of iterations but looking into the details, different tasks reach their optimal value after very different numbers of iterations. Additionally, the trade-off of heuristic accuracy and the time it takes to compute the heuristic value is different in each domain. The question then becomes whether to stop the heuristic computation with the current value or to spend more time on improving it by finding a better cost partitioning. Meta-reasoning like this is an active area in the planning community (e.g., Domshlak, Karpas, and Markovitch 2010; Tolpin et al. 2013; Barley, Franco, and Riddle 2014; O'Ceallaigh and Ruml 2015).

Answers to these problems could also be found in the integer programming community where subgradient algorithms have been the subject of intense research. Among other developments, improvements to the basic method are known in the following areas:

- reduction in the zig-zagging behavior. The method can make little progress if $g^{(t+1)} \approx -g^{(t)}$, or if the projection step moves the point almost back to where we started, so that $\lambda^{(t+1)} \approx \lambda^{(t)}$. Deflection techniques (Camerini, Fratta, and Maffioli 1975), where the direction is taken as a convex combination of the current gradient and the previous direction, can solve the first issue; and conditional subgradient techniques (Larsson, Patriksson, and Strömberg 1996), where the direction is projected to the tangent cone of Ω at λ, can solve the second. Both methods can be combined to get a zig-zagging free subgradient method (d'Antonio and Frangioni 2009).
- Improved step length update policies. A lot of computational investigation has been devoted to developing step length updates that not only converge in theory but that perform reasonably well also in practice (e.g., Polyak 1969; Bahiense, Maculan, and Sagastizábal 2002; Caprara, Fischetti, and Toth 1999).
- Improved update mechanisms. Instead of updating λ by following the subgradient no matter what, the step is taken only if it yields a sufficient improvement in the objective value. Otherwise a new direction is computed (Barahona and Anbil 2000; Bahiense, Maculan, and Sagastizábal 2002). This is similar to Bundle methods that distinguish between so called *serious steps* and *null steps*.
- Preserved multipliers. Reusing multipliers can speed up incremental computation and has an interesting analogy in planning: Reusing a multiplier corresponds to reusing a cost partitioning. A cost partitioning that is optimal for one state can be suboptimal but still good for another state, so the optimal partitioning does not have to be computed in each state. Potential heuristics are dual to operator-counting heuristics and typically not optimized in every state. Seipp, Pommerening, and Helmert (2015) use a diverse set of potential heuristics which can be thought of as using several multipliers. They also found that often a handful of cost partitionings (multipliers) are sufficient to get optimal values in all states, so seeding the method with previously used multipliers instead of starting from a fixed seed sounds promising.

We refer to the recent computational study by Frangioni, Gendron, and Gorgone (2017) for more details.

On the practical side, Lagrangian decomposition lends itself to parallelization as all subproblems are independent and can trivially be solved in parallel. We ran all our experiments in a single thread but parallelizing the algorithm would be a small step with a potentially big speedup.

Conclusion

We applied Lagrangian decomposition to operator-counting LPs and found that the Lagrangian multipliers directly correspond to partitioned cost functions. Using the subgradient method, we can find suboptimal cost partitionings that converge to optimal ones. In abstraction heuristics the subgradient has a clear interpretation as the number of times operators are used in a shortest plan.

We developed an algorithm that converges to the optimal non-negative cost partitioning of abstraction heuristics and can be computed without an LP solver. In practice, the algorithm finds close-to-optimal solutions with a low number of iterations and can often be computed orders of magnitude faster than finding the optimal cost partitioning with an LP.

The relationship of cost partitioning to subgradients opens up a large field of techniques investigated by the operations research community that can make cost partitioning more practical. Investigating stopping conditions, gradient scaling and deflection techniques sounds particularly promising.

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). Initial discussions started at the Dagstuhl seminar 18071 on *Planning and Operations Research*.

References

Anstreicher, K. M., and Wolsey, L. A. 2009. Two "well-known" properties of subgradient optimization. *Mathematical Programming* 120(1):213–220.

Bahiense, L.; Maculan, N.; and Sagastizábal, C. 2002. The volume algorithm revisited: relation with bundle methods. *Mathematical Programming* 94(1):41–69.

Barahona, F., and Anbil, R. 2000. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming* 87(3):385–399.

Barley, M. W.; Franco, S.; and Riddle, P. J. 2014. Overcoming the utility problem in heuristic generation: Why time matters. 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)*, 38–46. AAAI Press.

Camerini, P. M.; Fratta, L.; and Maffioli, F. 1975. On improving relaxation methods by modified gradient techniques. In *Nondifferentiable Optimization*. Springer Berlin Heidelberg. 26–34.

Caprara, A.; Fischetti, M.; and Toth, P. 1999. A heuristic method for set covering problem. *Operations Research* 47(5):730–743.

d'Antonio, G., and Frangioni, A. 2009. Convergence analysis of deflected conditional approximate subgradient methods. *SIAM Journal on Optimization* 20(1):357–386.

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

Domshlak, C.; Karpas, E.; and Markovitch, S. 2010. To max or not to max: Online learning for speeding up optimal planning. In Fox, M., and Poole, D., eds., *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*. AAAI Press.

du Merle, O.; Goffin, J.-L.; and Vial, J.-P. 1998. On improvements to the analytic center cutting plane method. *Computational Optimization and Applications* 11(1):37–52.

Frangioni, A.; Gendron, B.; and Gorgone, E. 2017. On the computational efficiency of subgradient methods: a case study with Lagrangian bounds. *Mathematical Programming Computation* 9(4):573–604.

Geoffrion, A. M. 1974. Lagrangian relaxation for integer programming. *Mathematical Programming* 2:82–114.

Guignard, M., and Kim, S. 1987. Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Mathematical Programming* 39(2):215–228.

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

Hiriart-Urruty, J.-B., and Lemaréchal, C. 1993. *Convex Analysis and Minimization Algorithms – II Advanced Theory and Bundle Methods*. Springer-Verlag.

Katz, M., and Domshlak, C. 2007. Structural patterns heuristics: Basic idea and concrete instance. In *ICAPS 2007 Workshop on Heuristics for Domain-Independent Planning: Progress, Ideas, Limitations, Challenges*.

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

Larsson, T.; Patriksson, M.; and Strömberg, A. 1996. Conditional subgradient optimization – theory and applications. *European Journal of Operational Research* 88(2):382–403.

O'Ceallaigh, D., and Ruml, W. 2015. Metareasoning in real-time heuristic search. In Lelis, L., and Stern, R., eds., *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)*, 87–95. AAAI Press.

Polyak, B. 1969. Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics* 9(3):14–29.

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.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From non-negative to general operator cost partitioning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3335–3341. AAAI Press.

Pommerening, F.; Helmert, M.; and Bonet, B. 2017. Abstraction heuristics, cost partitioning and network flows. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 228–232. AAAI Press.

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

Seipp, J.; Keller, T.; and Helmert, M. 2017a. A comparison of cost partitioning algorithms for optimal classical planning. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 259–268. AAAI Press.

Seipp, J.; Keller, T.; and Helmert, M. 2017b. 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)*, 3651–3657. AAAI Press.

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New optimization functions for potential heuristics. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 193–201. AAAI Press.

Shor, N. Z. 1985. *Minimization methods for nondifferentiable functions*. Springer-Verlag.

Tolpin, D.; Shimony, S. E.; Felner, A.; and Karpas, E. 2013. Towards rational deployment of multiple heuristics in A*. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 674–680. AAAI Press.

Yang, F.; Culberson, J.; Holte, R.; Zahavi, U.; and Felner, A. 2008. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research* 32:631–662.