Lagrangian Decomposition for Classical Planning (Extended Abstract)[†]

Florian Pommerening[1*] , **Gabriele Röger**[1] , **Malte Helmert**[1] ,
Hadrien Cambazard[2] , **Louis-Martin Rousseau**[3] and **Domenico Salvagnin**[4]
[1]University of Basel, Basel, Switzerland
[2]Univ. Grenoble Alpes, Grenoble INP, G-SCOP, 38000 Grenoble, France
[3]Polytechnique Montreal, Montreal, Canada
[4]University of Padua, Padua, Italy
{florian.pommerening,gabriele.roeger,malte.helmert}@unibas.ch, hadrien.cambazard@grenoble-inp.fr,
louis-martin.rousseau@polymtl.ca, 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. We analyze the application of Lagrangian decomposition, a classical tool in mathematical programming, to cost partitioning of operator-counting heuristics. This allows us to view the computation as an iterative process that can be seeded with any cost partitioning and that improves over time. In the case of non-negative cost partitioning of abstraction heuristics the computation reduces to independent shortest path problems and does not require an LP solver.

1 Introduction

The goal of optimal classical planning is to find a cheapest sequence of actions that transform a given initial state into a state satisfying some goal conditions. Each action transforms the world in a deterministic way and is associated with a cost. An optimal plan is a sequence of actions that reaches a goal with minimal total cost. A typical approach to finding such plans is heuristic search (e.g., with A^*) in the state space implicitly defined by the initial state, actions, and goal conditions. To guarantee optimality A^* requires an admissible heuristic function, i.e., a function that maps world states to estimates for the distance to the closest goal state without overestimating this distance. For example, abstraction heuristics homomorphically map the state space to an abstract state space that is small enough to be constructed explicitly. Every plan in the original state space remains a plan in the abstract state space, so the cost of a shortest abstract plan can be used as an admissible heuristic. The most common example of abstraction heuristics are pattern database (PDB) heuristics [Edelkamp, 2001] that project the factored state space to a subset of its factors (called variables).

Multiple admissible heuristics can be combined admissibly by using their maximum but the maximum of a large number of weak heuristics remains weak. Cost partitioning [Katz and Domshlak, 2010; Pommerening *et al.*, 2015] splits up the cost function and evaluates each involved heuristic under a reduced cost function to make their sum admissible. The optimal way of splitting up the cost function can be computed for abstraction heuristics in time polynomial in the size of the abstract state spaces. Even with small PDB heuristics that consider only up to three variables each, cost partitioning can produce near-perfect heuristic values [Pommerening, 2017]. While the computation of these values is polynomial in the size of abstractions, it requires solving a large linear program (LP) which is still prohibitively expensive to compute in many cases.

Operator-counting heuristics [Pommerening *et al.*, 2014] offer an alternative way of admissibly combining admissible heuristics. Abstraction heuristics and many other heuristics can be expressed with an LP over variables that express how often an operator is used. Each such heuristic can be written as linear constraints expressing a necessary property of a plan, so minimizing the total cost subject these constraints is an admissible heuristic. Multiple such heuristics can be combined by using all their constraints in a single LP. The resulting heuristic combination is equivalent to the optimal cost partitioning of the operator-counting heuristics for the individual constraints. While the LPs for abstraction heuristics are more compact than their cost partitioning counterparts, their evaluation is still too expensive to handle a large number of component heuristics.

The LPs computed by operator-counting heuristics exhibit a structure that can be exploited by Lagrangian decomposition [Geoffrion, 1974; Guignard and Kim, 1987], a classical tool from mathematical programming. It splits a large LP with a block structure into subproblems that depend on a set of parameters called the Lagrangian multipliers. Solving each subproblem for an optimal value of the multipliers achieves the same value as the original LP. The problem thus decomposes into finding good values for the multipliers and optimal solutions for the subproblems under these multipliers. We show that in the context of operator-counting heuristics the multipliers correspond to cost functions for the individual (cost-partitioned) heuristics.

Rather than evaluating the monolithic LP other methods

*Contact Author
†This is an abridged version of a paper that won the best paper award at ICAPS 2019 [Pommerening *et al.*, 2019].

can be used on the decomposed problem. For example, subgradient optimization is an iterative process that can be used to optimize the Lagrangian multipliers. We show that in the case of operator-counting heuristics the subgradients are updates to the cost functions and have a clear and intuitive interpretation. In the special case of non-negative cost partitioning of abstraction heuristics, a subgradient approach on the Lagrangian decomposition corresponds to a simple algorithm. It repeatedly solves a shortest path problem in each abstract state space under a certain cost function and uses the solutions to update the cost functions. Each iteration produces a (suboptimal) cost partitioning, so the algorithm can be stopped at any time with the best known cost partitioning.

2 Background

Consider the mathematical program $\min_{x\in\mathbb{R}^n}\{c^\top x \mid Ax \geq b, x \in \mathbb{X}\}$ with a linear constraint $Ax \geq b$ (the *complicating constraint*) and an arbitrary constraint $x \in \mathbb{X}$. *Lagrangian relaxation* of the constraint $Ax \geq b$ introduces a penalty term $\lambda \in \mathbb{R}^m_{\geq 0}$ (where m is the number of rows in A) called the *Lagrangian multiplier*. The constraint is then relaxed to the objective function as $\phi(\lambda) = \min_{x\in\mathbb{R}^n}\{c^\top x + \lambda^\top(b - Ax) \mid x \in \mathbb{X}\}$. For every choice of values for λ, $\phi(\lambda)$ is a lower bound to the original problem, and finding the best lower bound is the *Lagrangian dual*: $\phi^* = \max_{\lambda\in\mathbb{R}^m_{\geq 0}} \phi(\lambda)$. The function $\phi(\lambda)$ is continuous and concave, so it can be optimized with any nondifferentiable optimization algorithm such as the subgradient method [Shor, 1985].

Lagrangian decomposition applies Lagrangian relaxation to programs that decompose into independent subproblems when the complicating constraints are removed. We are interested in an LP P that does not have complicating constraints but complicating variables x:

$$\begin{aligned} \min c^\top x \quad & \text{subject to} \\ & A_1 x + B_1 y_1 \geq b_1 \\ & \cdots \\ & A_k x + B_k y_k \geq b_k \\ & x, y_1, \ldots, y_k \geq 0 \end{aligned} \tag{1}$$

We can bring P into the required form by creating copies x_i of the variables x and adding constraints $x = x_i$ for $1 \leq i \leq k$. The resulting subproblems then are

$$\begin{aligned} \phi_i(\lambda_i) = \min \lambda_i^\top x_i \quad & \text{subject to} \\ & A_i x_i + B_i y_i \geq b_i \\ & x_i, y_i \geq 0 \end{aligned} \tag{2}$$

and the value of our original problem P is

$$\phi^* = \max \sum_{i=1}^k \phi_i(\lambda_i) \quad \text{subject to} \quad \sum_{i=1}^k \lambda_i \leq c. \tag{3}$$

For lack of space we refer to the original paper for the detailed derivation [Pommerening *et al.*, 2019].

We use the subgradient method to compute ϕ^*, which starts from any vector $\lambda^{(1)}$ and repeatedly updates it following a subgradient. In iteration t, the next value is computed as

$$\lambda^{(t+1)} = P_\Omega\left(\lambda^{(t)} + \eta(t) g^{(t)}\right) \tag{4}$$

where $g^{(t)}$ is a subgradient of ϕ at $\lambda^{(t)}$, $\eta(t) > 0$ is the step length at iteration t, and P_Ω is the projection to the space of vectors satisfying $\sum_{i=1}^k \lambda_i \leq c$. Under mild assumptions on the function η, the method converges to an optimal solution. In our decomposed problem, subgradients $g^{(t)}$ are concatenations of subgradients for each subproblem ϕ_i. From the definition of ϕ_i it can be shown that optimal solutions of the problems $\phi_i(\lambda^{(t)})$ are such subgradients.

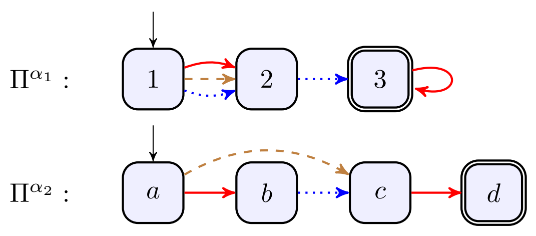
3 Application to Operator Counting

Operator-counting heuristics are computed by an LP of the form (1) where the variables x represent how often an action is used, the objective coefficients c correspond to the costs of each action, and each group of constraints $A_i x + B_i y_i \geq b_i$ corresponds to one component heuristic. The variables y_i are auxiliary variables that are not shared between constraints and can have different interpretations in different constraints. The heuristic is admissible if every constraint $A_i x + B_i y_i \geq b_i$ has a solution for every plan π where x_o is the number of times action o is used in π. Different heuristics can be expressed in this way. For example, abstraction heuristics are expressed by constraints that describe a shortest path problem in the abstract state space. As mentioned earlier, individual heuristics can be combined by optimizing the LP subject to the union of their constraints, and this computes their optimal cost partitioning.

Looking at the subproblems $\phi_i(\lambda_i)$ of the Lagrangian decomposition (2) we can see that they have the same form as an operator-counting heuristic for the single component heuristic i, but instead of the original cost function c, they use the cost function λ_i. In fact, they compute the component heuristic i under the modified cost function λ_i. The Lagrangian dual ϕ^* (3) then optimizes the way the cost functions λ_i are chosen in a way that maximizes the sum of the heuristic values while satisfying the cost partitioning constraint $\sum_{i=1}^k \lambda_i \leq c$ that guarantees that costs are partitioned.

Applying the subgradient method to this problem means that we start from any cost partitioning ($\lambda^{(1)}$) and continue to update it with subgradients according to equation (4). The subgradients correspond to optimal solutions of the subproblems (2), i.e., values for the operator-counting variables in an optimal solution of the component heuristic. For abstraction heuristics, we can specialize this algorithm further because the operator-counting variables in an optimal solution directly correspond to the number of times an operator is used in an optimal abstract 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. 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. Let $\mathit{occurrences}(o, \pi_i^{(t)})$ be the number of occurrences of o in $\pi_i^{(t)}$.
3. Follow the subgradient for a step with length $\eta(t)$, i.e., set $c_i^{(t+1)}(o) = \mathit{cost}_i^{(t)}(o) + \eta(t)\mathit{occurrences}(o, \pi_i^{(t)})$.



t		α_1 →	α_1 ⇢	α_1 ⋯→	α_2 →	α_2 ⇢	α_2 ⋯→
1	*cost*	0.5	0.5	0.5	0.5	0.5	0.5
	$\pi/cost(\pi)$	⟨→, ⋯→⟩/1			⟨⇢, →⟩/1		
	x^*	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
	$\pi/cost(\pi)$	⟨⇢, ⋯→⟩/1			⟨→, ⋯→, →⟩/1		
	x^*	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
	$\pi/cost(\pi)$	⟨→, ⋯→⟩/1			⟨⇢, →⟩/1.75		
	x^*	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
	$\pi/cost(\pi)$	⟨→, ⋯→⟩/1			⟨⇢, →⟩/$1.91\bar{6}$		
	x^*	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
	$\pi/cost(\pi)$	⟨→, ⋯→, →⟩/1			⟨→, ⋯→, →⟩/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.

4. Set $cost^{(t+1)} = P_\Omega(c_1^{(t+1)}, \dots, c_k^{(t+1)})$, where $\Omega = \{\langle c_1, \dots, c_k\rangle \in \mathbb{R}^{k|O|} \mid \sum_{i=1}^{k} c_i \leq cost\}$.

The final step projects cost functions into the space of cost partitionings. If all costs are non-negative, this can be done by setting $cost_i^{t+1}(o) = \max(c_i^{t+1}(o) - \delta^{t+1}(o), 0)$, where $\delta^{t+1}(o)$ is the smallest adjustment necessary to satisfy the cost partitioning constraint for o.

4 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 →, ⇢, and ⋯→ that all have a cost of 1. The cost partitioning is initialized to the uniform cost partitioning which assigns a cost of 0.5 to each operator 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 ⟨→, ⋯→⟩. In α_2 there is only one choice: ⟨⇢, →⟩.

With step length $\eta(1) = 1$, the cost of → is increased by 1 in both abstractions, while the cost of ⋯→ is only increased in α_1 and that of ⇢ only in α_2. The resulting cost functions *cost′* no longer satisfy the cost partitioning constraint and have to be projected back into the space of non-negative cost partitionings. For → 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 (⟨⇢, ⋯→⟩ and ⟨→, ⋯→, →⟩). 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 ⇢ and ⋯→ 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 ⋯→ by 0.5 as well but increase the cost of → by 1 because it is used twice. The projection decreases the costs of → and ⋯→ by 0.5 and the costs of ⇢ by 0.25 in both abstractions.

After this step, the two shortest plans (⟨→, ⋯→⟩ and ⟨⇢, →⟩) 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 improves 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 ⇢ cannot be reduced by the same amount in α_1 and α_2, so its cost is reduced to 0 in α_1 and to 1 in α_2.

The final cost partitioning is optimal in this case and has a heuristic value of 3. There are multiple shortest plans under these cost functions, but note that there are shortest plans that use each operator the same number of times in both abstractions. 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 $\mathbf{0}$ is a subgradient and we have reached an optimal value.

5 Experiments

We implemented the subgradient algorithm in the Fast Downward planning system [Helmert, 2006] and evaluated it on instances from the international planning competitions (IPC 1998–2018). We refer to the original paper for additional details and results. The results we discuss here use all projections to 1 or 2 variables of the planning task that are relevant in the context of non-negative cost partitioning. We limit time to 300 s and memory to 2 GiB for each instance. As a step length function, we use $\eta(t) = 1/t$, which guarantees convergence to an optimal solution. We tested four suboptimal cost partitioning methods to seed the algorithm: uniform, opportunistic uniform, greedy zero-one, and saturated cost partitioning [Seipp *et al.*, 2017]. The latter three depend on an order of the abstractions. We tried both a random order and and order that was greedily improved for 100 s.

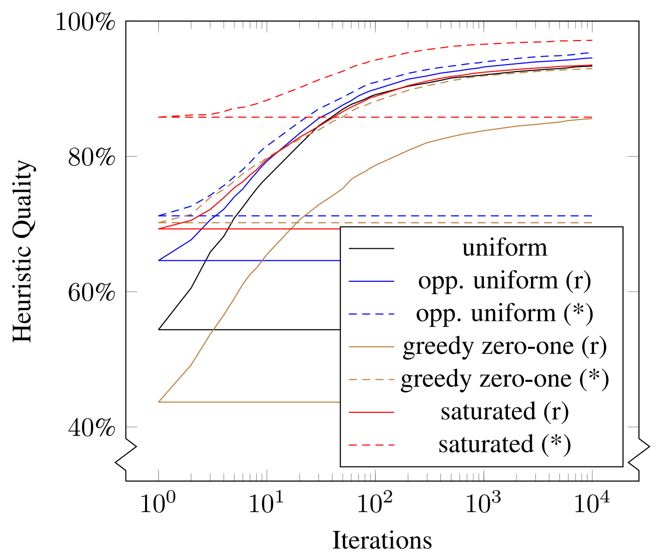


Figure 2: Heuristic quality of different suboptimal cost partitioning methods measured as the geometric mean of the ratio $h^C(s)/h^{C^*}(s)$. Order-dependent methods are used with random (r) or improved (*) orders. In each case, the horizontal line shows the quality of the cost partitioning method while 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 2 shows how the heuristic quality of the initial state (compared to optimal cost partitioning) improves with iterations of the subgradient algorithm. In all cases, the quality improves steeply in the first few iterations. In the first 200 iterations most methods reach a quality of at least 90%, surpassing our best baseline method (saturated cost partitioning with an improved order). The results also show that starting from a higher quality seed generally leads to better results but all seeds can be improved to a high quality.

One source of suboptimality were tasks with high action costs such as from the domain ParcPrinter. In these instances, actions can have costs on the order of 10^6 but plans are usually short, using most operators at most once. Combined with a step length function of $\eta(t) = 1/t$ this means that the cost functions change by less than 1 in most steps, and it takes many iterations to significantly modify a cost partitioning.

The above implementation does not require an LP solver and uses Dijkstra's [1959] algorithm to solve the shortest path problems. We compared this to a more straight-forward implementation of Lagrangian decomposition that uses CPLEX 12.8 to solve the subproblems. The abstract state spaces of our abstractions are very small ($< 10^5$ states), so the overhead of an LP solver over a specialized algorithm is high. In our experiments, the LP solver was 10–200 times slower.

Finally, Figure 3 compares the time to compute the first 200 iterations of the subgradient algorithm to the time to evaluate the monolithic LP that computes the optimal non-negative cost partitioning. The time for the subgradient method scales linearly with the number of iterations, so there is a trade-off between accuracy and speed. For easier instances where the

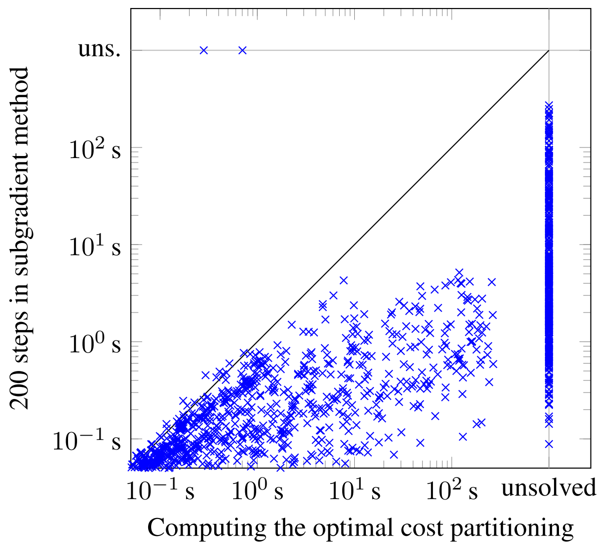


Figure 3: Time required to compute the heuristic value of the initial state using either 200 iterations of the subgradient method or solving the monolithic cost partitioning LP.

optimum is reached in less than 200 iterations, we can see that computing the monolithic LP is faster, but in general the results show that an accuracy of 90% can be reached in a fraction of the time. There are also many cases where the monolithic LP exceeded the resource limits but the subgradient algorithm could still compute a value.

6 Conclusion

We applied Lagrangian decomposition to operator-counting heuristics and found that the multipliers correspond to partitioned cost functions. We developed a general anytime algorithm that converges to an optimal cost partitioning and a specialization for non-negative cost partitioning of abstraction heuristics that does not rely on an LP solver.

The interpretation with Lagrangian decomposition opens operator-counting heuristics to all techniques developed in the area of nondifferentiable optimization. We used the subgradient method with straight-forward choices for gradient, step length, and stopping condition. More elaborate choices are possible, and there is a rich literature in the area with many possible extensions. We discuss this in more detail in the original paper.

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

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

[Edelkamp, 2001] Stefan Edelkamp. Planning with pattern databases. In Amedeo Cesta and Daniel Borrajo, editors, *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, pages 84–90. AAAI Press, 2001.

[Geoffrion, 1974] Arthur M. Geoffrion. Lagrangean relaxation for integer programming. *Mathematical Programming*, 2:82–114, 1974.

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

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

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

[Pommerening *et al.*, 2014] Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. LP-based heuristics for cost-optimal planning. In Steve Chien, Alan Fern, Wheeler Ruml, and Minh Do, editors, *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, pages 226–234. AAAI Press, 2014.

[Pommerening *et al.*, 2015] Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From non-negative to general operator cost partitioning. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, pages 3335–3341. AAAI Press, 2015.

[Pommerening *et al.*, 2019] Florian Pommerening, Gabriele Röger, Malte Helmert, Hadrien Cambazard, Louis-Martin Rousseau, and Domenico Salvagnin. Lagrangian decomposition for optimal cost partitioning. In Nir Lipovetzky, Eva Onaindia, and David E. Smith, editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, pages 338–347. AAAI Press, 2019.

[Pommerening, 2017] Florian Pommerening. *New Perspectives on Cost Partitioning for Optimal Classical Planning*. PhD thesis, University of Basel, 2017.

[Seipp *et al.*, 2017] Jendrik Seipp, Thomas Keller, and Malte Helmert. A comparison of cost partitioning algorithms for optimal classical planning. In Laura Barbulescu, Jeremy Frank, Mausam, and Stephen F. Smith, editors, *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, pages 259–268. AAAI Press, 2017.

[Shor, 1985] Naum Zuselevich Shor. *Minimization methods for non-differentiable functions*, volume 3. Springer-Verlag Berlin Heidelberg, 1985.