New Refinement Strategies for Cartesian Abstractions

David Speck,[1] **Jendrik Seipp**[2]

[1]University of Freiburg, Freiburg, Germany
[2]Linköping University, Linköping, Sweden
speckd@informatik.uni-freiburg.de, jendrik.seipp@liu.se

Abstract

Cartesian counterexample-guided abstraction refinement (CEGAR) yields strong heuristics for optimal classical planning. CEGAR repeatedly finds counterexamples, i.e., abstract plans that fail for the concrete task. Although there are usually many such abstract plans to choose from, the refinement strategy from previous work is to choose an arbitrary optimal one. In this work, we show that an informed refinement strategy is critical in theory and practice. We demonstrate that it is possible to execute all optimal abstract plans in the concrete task simultaneously, and present methods to minimize the time and number of refinement steps until we find a concrete solution. The resulting algorithm solves more tasks than the previous state of the art for Cartesian CEGAR, both during refinement and when used as a heuristic in an A* search.

Introduction

Optimal classical planning is the problem of finding a cheapest plan (sequence of actions) for a given planning task that leads from an initial situation to a desired goal situation. A prominent solution method for this problem is A* (Hart, Nilsson, and Raphael 1968) with an admissible heuristic (Pearl 1984). Cartesian abstractions (Ball, Podelski, and Rajamani 2001) represent a smaller and simpler version of the given concrete task and provide a way to obtain admissible heuristics by either considering a single abstraction or combining multiple abstractions (Katz and Domshlak 2008; Yang et al. 2008; Pommerening et al. 2015; Seipp, Keller, and Helmert 2020; Drexler, Seipp, and Speck 2021).

Counterexample-guided abstraction refinement (CEGAR) is one way to iteratively generate such Cartesian abstractions with increasing accuracy (Clarke et al. 2000). In planning, Cartesian CEGAR (Seipp and Helmert 2018; Seipp, von Allmen, and Helmert 2020) repeatedly finds optimal counterexamples, i.e., optimal abstract plans that fail for the concrete task. Based on the selected abstract plan, the abstraction is refined so that the detected flaw no longer occurs. Although there are usually many different optimal abstract plans in an abstraction, the refinement strategy from previous work is to choose an arbitrary one. We show that selecting the optimal abstract plan that comes closest to solving the concrete task can lead to arbitrarily fewer refinements.

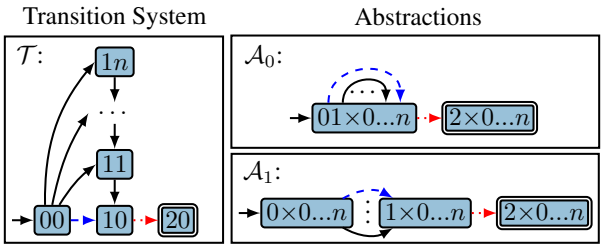


Figure 1: The concrete transition system (left) of a planning task for which our new refinement strategies yield arbitrarily smaller abstractions (right) until a concrete plan is found. The task contains a dashed blue operator, a dotted red operator and multiple solid black operators.

For example, consider abstraction $\mathcal{A}_1$ of the concrete transition system $\mathcal{T}$, both shown in Figure 1. Applying the dashed blue and then the dotted red operator is an optimal concrete plan, while any plan that starts with one of the solid black operators is a counterexample that leads to (in the worst case arbitrarily many) unneeded refinements.

In this paper, we investigate new refinement strategies for Cartesian CEGAR that consider *all* optimal abstract plans, and show that these new strategies theoretically and empirically improve the state of the art for Cartesian abstractions.

Background

A *transition system* $\mathcal{T}$ is a labeled and directed graph with a finite set of *states* $S(\mathcal{T})$, a finite set of *labels* $L(\mathcal{T})$ together with a *cost* function $c(\mathcal{T}) : L(\mathcal{T}) \to \mathbb{R}_0^+$, a finite set of labeled *transitions* $T \subseteq S(\mathcal{T}) \times L(\mathcal{T}) \times S(\mathcal{T})$, an *initial state* $s_0(\mathcal{T}) \in S(\mathcal{T})$, and a set of *goal states* $S_\star(\mathcal{T}) \subseteq S(\mathcal{T})$. A *path* τ from s_0 to s_n is an interleaved sequence of states and labels $\langle s_0, l_1, s_1, \dots s_{n-1}, l_n, s_n \rangle$ with $\langle s_{i-1}, l_i, s_i \rangle \in T(\mathcal{T})$. We call τ a *goal path* if it is a path from the initial state $s_0(\mathcal{T})$ to a goal state $s_\star \in S_\star(\mathcal{T})$, and the sequence of labels $\pi = \langle l_1, \dots, l_n \rangle$ of a goal path is called a *plan*. The *cost* of a path τ is the cumulative cost of the induced plan π, i.e., $c(\tau) = c(\pi) = \sum_{i=1}^{n} c(\mathcal{T})(l_i)$. A path or plan is called *optimal* if there is no cheaper one. We call transitions and states *f-optimal* if they are part of an optimal goal path. We define $h(s)$ of a state s as the minimum cost over all optimal paths from s and set $h(s) = \infty$ if there is no such path.

A SAS$^+$ *planning task* (Bäckström and Nebel 1995) is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ consisting of a finite set of *variables* $\mathcal{V}$, where each variable $v \in \mathcal{V}$ is associated with a finite *domain* $dom(v)$; a finite set of *operators* $\mathcal{O}$, where each operator $o \in \mathcal{O}$ is associated with *costs* $c(o) \in \mathbb{R}_0^+$; an *initial state* s_0; and a partial state called *goal condition* $s_\star$, specifying the set of *goal states* $S_\star \subseteq S(\Pi)$. A *partial state* s is a mapping of a subset of variables $\mathrm{vars}(s) \subseteq \mathcal{V}$ to values in their domains. We call s a *state* if $\mathrm{vars}(s) = \mathcal{V}$ and refer to all possible states by $S(\Pi)$. An operator $o \in \mathcal{O}$ consists of the partial states pre_o and eff_o and is *applicable* in a state s iff $\mathrm{pre}_o \subseteq s$. Applying o in s yields the state $s[\![o]\!]$ with $s[\![o]\!](v) = \mathrm{eff}_o(v)$ if $v \in \mathrm{vars}(\mathrm{eff}_o)$ and $s[\![o]\!](v) = s(v)$ otherwise. A planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ induces a *concrete transition system* $\mathcal{T}$ with $S(\mathcal{T}) = S(\Pi)$, $L(\mathcal{T}) = \mathcal{O}$, $s_0(\mathcal{T}) = s_0$, $T(\mathcal{T}) = \{\langle s, l, s[\![o]\!] \rangle \mid s \in S(\Pi), o \in \mathcal{O}, \mathrm{pre}_o \subseteq s\}$, and $S_\star(\mathcal{T}) = S_\star(\Pi)$.[1]

A set of states c is called *Cartesian* if it has the form $D_1 \times \cdots \times D_{|\mathcal{V}|}$, where $D_i \subseteq dom(v_i)$. By $dom(v_i, c) = D_i \subseteq dom(v_i)$ we denote the set of values that variable v_i can have in c. An *abstraction* $\mathcal{A}$ of a concrete transition system $\mathcal{T}$ ignores some distinctions between states to obtain a smaller and simpler version of $\mathcal{T}$, while preserving all transitions (Helmert, Haslum, and Hoffmann 2007). The *(abstract) states* $S(\mathcal{A})$ of an abstraction $\mathcal{A}$ are the equivalence classes of an equivalence relation on $S(\mathcal{T})$. We consider a special kind of abstraction, *Cartesian abstractions* (Seipp and Helmert 2018), where all abstract states are Cartesian sets. We write $[s] \in S(\mathcal{A})$ for the unique abstract state that subsumes the concrete state $s \in S(\mathcal{T})$.

Counterexample-Guided Abstraction Refinement

Counterexample-guided abstraction refinement (CEGAR) is a method for incrementally computing more informative Cartesian abstractions for the concrete transition system $\mathcal{T}$ of a planning task Π (Seipp and Helmert 2013, 2018). As a preprocessing step, we separate all goal states from non-goal states by splitting all facts $(v, d) \in s_\star$ from the (always unique) goal state. Then CEGAR iteratively refines the abstraction $\mathcal{A}$ by searching for an optimal abstract plan π in $\mathcal{A}$, checking whether π is also a concrete plan for Π, and repairing the flaw of π in $\mathcal{A}$ if necessary.

The induced plan π of an optimal abstract goal path $\tau = \langle a_0, o_1, a_1, \ldots, a_{n-1}, o_n, a_n \rangle$ is either a concrete solution for $\mathcal{T}$ or yields a *flaw* $\varphi = \langle s, c \rangle$ which is a pair of a concrete state $s \in S(\mathcal{T})$ and a Cartesian set c, where either (i) $s = s_0[\![o_1]\!] \ldots [\![o_{i-1}]\!]$ is the first concrete state in which an operator o_i is not applicable and c is the set of concrete states in $[s] = a_i$ in which o_i is applicable, or (ii) $s = s_0[\![o_1]\!] \ldots [\![o_{i-1}]\!]$ is the first state in which o_i is applicable but $[s[\![o_i]\!]] \neq a_{i+1}$ and c is the set of concrete states in $[s] = a_i$ from which we can reach a_{i+1} by applying o_i.

By $\Phi(\mathcal{A})$ we denote the *set of all flaws* of an abstraction $\mathcal{A}$. Given a flaw $\varphi = \langle s, c \rangle$, we *repair* it by splitting $[s]$ into two abstract states d and e with $s \in d$ and $c \subseteq e$. To do this, we select a single variable v such that $s(v) \notin dom(v, c)$ and split $dom(v, [s])$ into $dom(v, d)$ and $dom(v, e)$. Often several different *splits* are possible by choosing different variables v to split for and by choosing how to split the corresponding domain $dom(v, [s])$.

Example 1. *Consider a family of unit-cost planning tasks Π_n with two variables $\mathcal{V} = \{v_1, v_2\}$, where $dom(v_1) = \{0, 1, 2\}$ and $dom(v_2) = \{0, \ldots, n\}$. Figure 1 shows the transition system $\mathcal{T}$ of Π_n, where each edge represents a distinct operator, yielding $2n + 2$ operators in total. To explain CEGAR, we consider the task Π_0, where the domain of variable v_1 is the singleton set $\{0\}$. CEGAR starts with the abstraction $\mathcal{A}_0$ consisting of a single non-goal state and one goal state (Figure 1). The cheapest plan in $\mathcal{A}_0$ is $\langle \mathrm{red} \rangle$ with $\mathrm{pre}_{\mathrm{red}} = \{(v_1, 1), (v_2, 0)\}$ and $\mathrm{eff}_{\mathrm{red}} = \{(v_1, 2), (v_2, 0)\}$ leading from state 10 to 20. Operator red is not applicable in the initial state 00 of $\mathcal{T}$, so the domain of v_1 needs to be split, resulting in $\mathcal{A}_1$. In $\mathcal{A}_1$, the optimal plan $\langle \mathrm{blue}, \mathrm{red} \rangle$ is a concrete solution and is therefore returned by CEGAR.*

For a more detailed account of CEGAR for planning, we refer the reader to Seipp and Helmert (2018).

New Refinement Strategies

Although there are usually many optimal abstract plans to choose from in a refinement step of CEGAR, the refinement strategy from previous work only considers a single arbitrary one. In this section, we introduce a method to find all flaws by executing all optimal abstract plans in the concrete task simultaneously. This allows us to compare different flaw selection strategies, for which we show that it can be arbitrarily better to select a flaw closer to a goal state than to choose the first encountered flaw. The second advantage of knowing all flaws in the current abstraction is that we can consider all flaws in an abstract state a when determining how best to split a. We call the combination of a flaw selection strategy and a split selection strategy a *refinement strategy*.

Finding All Flaws

Given an abstraction $\mathcal{A}$, we perform a depth-first search (called *flaw search*) in the concrete transition system $\mathcal{T}$, starting at s_0, but consider only f-optimal transitions of $\mathcal{A}$. In other words, for a state s, we consider not all successors $\{s[\![o]\!] \mid o \in \mathcal{O}, \mathrm{pre}_o \subseteq s\}$, but only those that have an f-optimal transition $\langle a, o, b \rangle$ in $\mathcal{A}$ such that $[s] = a$ and $[s[\![o]\!]] = b$.

To collect all flaws while running the flaw search, we make some further adaptations. When expanding a state s, we retrieve all f-optimal abstract transitions T^* starting in abstract state $[s]$. Then for each $\langle [s], o, b \rangle \in T^*$, we check whether o is applicable in s. If not, we have found a flaw $\langle s, c \rangle$ where c is the set of concrete states in $[s]$ in which o is applicable. If o is applicable in s but $[s[\![o]\!]] \neq b$, we have found a flaw $\langle s, c \rangle$, where c is the set of concrete states in $[s]$ from which we can reach b via o. The flaw search terminates when we expand a goal state and thus have found a concrete solution, or when the open list is empty and thus all flaws $\Phi(\mathcal{A})$ have been found.

The following proposition holds because we execute all optimal abstract plans in the concrete task.

[1] We support zero-cost operators by assigning sufficiently small positive costs ε to the corresponding transitions.

Proposition 1. *The flaw search returns either an optimal concrete plan or the set of all flaws for a given abstraction.*

Flaw Selection Strategies

Given the set of all flaws $\Phi(\mathcal{A})$ of an abstraction, the question is which flaws to repair. Any refinement is guaranteed to make the abstraction more accurate, but as Example 1 shows, some refinements will lead to finding a concrete solution faster than others. Previous work simply always refined the abstraction based on an arbitrary optimal abstract plan. We show below that it is much preferable to consider an optimal abstract plan that breaks closest to a goal state in the concrete transition system.

To evaluate this idea, we introduce two new flaw selection strategies, MINH and MAXH, which both consider all optimal abstract plans using the flaw search presented above. Both strategies select a flaw $\varphi = \langle s, c \rangle \in \Phi(\mathcal{A})$ based on the h value of the abstract state $[s] \in S(\mathcal{A})$ in which φ occurs. MINH returns a concrete solution if one exists, and selects a flaw with the lowest h value otherwise. MAXH selects a flaw with the highest h value if a flaw exists, and returns a concrete solution otherwise. Thus, MINH and MAXH prefer plans that break late and early, respectively. We use MAXH as a worst-case strategy that leads to refining the abstraction until all optimal abstract plans are valid concrete solutions. Additionally, we consider the FIRST strategy from previous work, which selects an arbitrary optimal abstract plan π, returns π if it works for the concrete task, or selects the first flaw found for π (Seipp and Helmert 2018).

Batch Refinements. Even if MINH needs fewer refinements than other strategies, we will see below that searching for all flaws $\Phi(\mathcal{A})$ in each refinement step is too costly if we only split a single state afterwards. To amortize the overhead of the flaw search, we introduce the BATCH flaw selection strategy that selects multiple flaws at once. BATCH runs the flaw search and returns a concrete plan if the flaw search found one. Otherwise, BATCH iteratively repairs the flaw with the lowest h value. Repairing a flaw changes the abstraction, which may affect the other flaws found. Therefore, after each refinement, we need to check if the h values of the remaining flaws have changed. If the h value of a flaw has not changed, this flaw is still part of an optimal abstract plan and we still need to repair it. Otherwise, we discard it. The BATCH strategy continues this process until all flaws in $\Phi(\mathcal{A})$ have been repaired or discarded. Repairing the flaws in ascending order of their h value as a batch is possible, because repairing a flaw refines the abstraction and therefore the h values in the abstraction can only increase. Thus, repairing a flaw can never decrease the h values of the flaws considered later, but it can increase their h values, in which case they are no longer part of an optimal abstract plan and hence can be discarded. Refining flaws in a batch comes at the cost of possibly repairing flaws with higher h value than necessary, because we may not yet know about flaws with smaller h value, or about an abstract plan that works for the concrete task.

The new flaw selection strategies MINH and BATCH, which prefer flaws near the goal, i.e., with a small h value, can lead to requiring arbitrarily fewer refinement steps.

Theorem 1. *Using the* FIRST *or* MAXH *flaw selection strategies can lead to arbitrarily larger abstractions until a concrete solution is found, compared to using the* MINH *or* BATCH *strategies.*

Proof. Consider the family of planning tasks Π_n described in Example 1 and visualized in Figure 1. Regardless of the choice of $n \in \mathbb{N}_0$, there is only one optimal abstract plan and hence one flaw in the initial abstraction $\mathcal{A}_0$. We obtain the abstraction $\mathcal{A}_1$, which has 3 states and n different optimal abstract plans, but only $\pi = \langle \text{blue}, \text{red} \rangle$ is a concrete plan. MINH and BATCH directly return π, since they consider all abstract plans. MAXH and possibly FIRST consider one of the n abstract plans $\pi' = \langle \text{black}, \text{red} \rangle$ which leads to splitting the abstract state $\{1\} \times \{0, \dots, n\}$. This can occur n times in total, separating each value of v_2 until the abstraction matches $\mathcal{T}$ with $n + 3$ states. At this point, CEGAR returns the optimal plan π. □

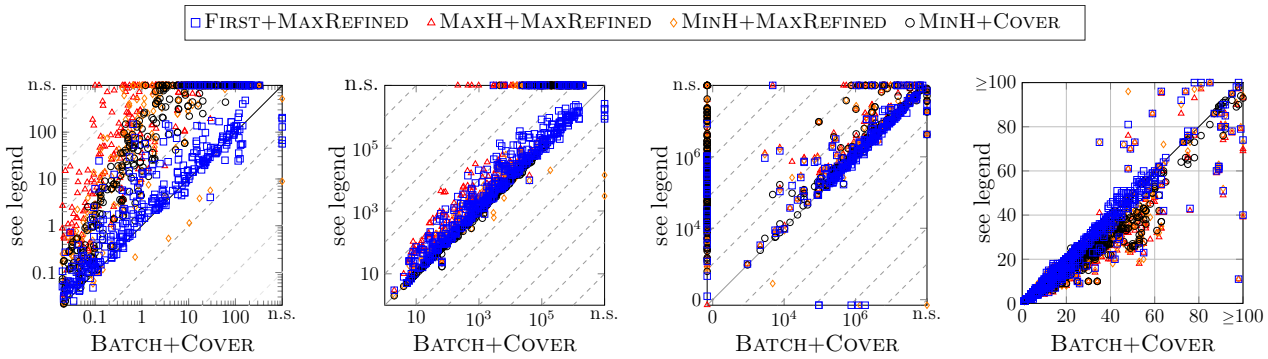
Split Selection Strategies

Given a flaw $\varphi = \langle s, c \rangle$, there are usually many different ways of how to split the abstract state $[s]$ into two new abstract states, all of which repair φ. The main choice point is choosing the variable v whose domain $dom(v, [s])$ is to be split but there are usually also several possibilities of how to partition $dom(v, [s])$. Seipp and Helmert (2018) use the MAXREFINED split selection strategy that selects the variable v which maximizes the fraction $\frac{dom(v,[s])}{dom(v)}$ among all variables for which splits are feasible.

By running the flaw search, we obtain all flaws that occur in an abstract state, instead of only a single one as in previous work. This allows us to define the COVER split selection strategy, which splits an abstract state in a way that maximizes the number of repaired flaws by a single refinement. COVER considers all possible splits for each flaw, counts how many other flaws they repair, and chooses the split that addresses the most flaws at once.

Experiments

We implemented the new refinement strategies (each consisting of a flaw and split selection strategy) for Cartesian CEGAR in the Scorpion planner (Seipp 2018), which builds on Fast Downward (Helmert 2006). We evaluate the refinement strategies and compare them to the previous state of the art (FIRST+MAXREFINED) on all 1827 tasks (65 domains) from the optimal tracks of the International Planning Competitions 1998–2018 using Lab (Seipp et al. 2017). All of our benchmarks, source code and experiment data are available online (Speck and Seipp 2022). We use an overall time limit of 30 minutes and a memory limit of 4 GB, refine a single abstraction and use the resulting h^{CEGAR} heuristic in an A^* search. We stop the refinement when we find a concrete solution, reach the refinement time limit of 15 minutes, or use more than 3.5 GB for the abstraction or the flaw search. For space reasons, we only present results for an interesting subset of combinations of flaw selection and split selection strategies.



(a) CEGAR runtime (seconds) (b) Abstraction size (#states) (c) Expansions before last f layer (d) Final $h^{\text{CEGAR}}(s_0)$ value

Figure 2: Comparison of BATCH+COVER to other refinement strategies for creating h^{CEGAR} and using it in an A* search. Tasks that are not solved within the resource limits appear on "n.s." axes.

	MAXREFINED			COVER	
Strategy	FIRST	MAXH	MINH	MINH	BATCH
Refinement	499	345	382	393	**534**
Refinement + A*	802	780	792	797	**812**

Table 1: Number of solved tasks by different refinement strategies. The first row shows how many tasks are solved during refinement.

Table 1 shows the number of solved tasks by different refinement strategies. Especially interesting is the first row, which shows how many tasks are solved during refinement. It reveals that it is too expensive to collect all flaws in each refinement step and repair only a single one of them, as the three strategies in the middle do. Instead, it is preferable to consider an arbitrary optimal path and repair only its first flaw (FIRST+MAXREFINED), as done in previous work. However, we solve the most tasks during refinement, if we compute all flaws, use batch refinement and try to cover several flaws of each abstract state simultaneously (BATCH+COVER). This strategy solves many more tasks than the previous best strategy, both during refinement (534 vs. 499 tasks) and even overall (812 vs. 802 tasks).

Figure 2 compares our strongest refinement strategy, BATCH+COVER, with the other four strategies from Table 1 in more detail. For the tasks solved during the refinement, Figures 2a and 2b present the time to create h^{CEGAR} and the resulting abstraction size. The plots reveal that BATCH+COVER requires fewer refinements, often by more than one order of magnitude (Figure 2b), and solves tasks faster (Figure 2a) than the previous state of the art and the other strategies. Figures 2c and 2d show the heuristic accuracy of the h^{CEGAR} heuristics obtained with the different refinement strategies. Since BATCH+COVER makes better refinements than the other strategies, the resulting heuristic is almost always more accurate and is perfect for many more tasks than the heuristics computed by the other strategies ($x = 0$ in Figure 2c). Similarly, for the vast majority of tasks, BATCH+COVER is able to prove higher lower bounds on solution cost than the other strategies (Figure 2d).

Related Work

In the context of CEGAR for pattern selection, Rovner, Sievers, and Helmert (2019) study so-called *wildcard plans*, i.e., abstract plans that share the same sequence of abstract states but may use different operators between states. Wildcard plans usually consider more than one abstract plan, but they are less general than our approach of considering all optimal plans. Thus, only considering a single wildcard plan can still lead to making many unnecessary refinements when a different wildcard plan already contains a concrete plan.

Top-k and top-quality planning is concerned with finding a set of best plans (Speck, Mattmüller, and Nebel 2020; Katz, Sohrabi, and Udrea 2020). While in these scenarios all plans are generated explicitly, in our case we are only interested in which concrete states are reachable by optimal abstract plans. Thus, we do not generate the actual plans, which would require much more effort, since all the possibilities of how a state can be reached have to be stored.

Apart from MAXREFINED there are other split selection strategies for a single flaw (Seipp and Helmert 2018). Among those strategies MAXREFINED is the preferable option. However, our analyses show that for finding concrete solutions quickly, selecting which state to split is more important than choosing how to split it.

Conclusions

We introduced new refinement strategies for Cartesian CEGAR, that differ in how they select flaws and how they compute splits that repair the flaws. Our algorithm for executing all optimal abstract plans in the concrete task simultaneously is able to obtain all flaws of an abstraction. Our theoretical and empirical analyses show that it is beneficial to 1) prefer refining the abstraction near the goal, 2) refine an abstract state such that the number of repaired flaws is maximized, and 3) refine several abstract states at once. The resulting algorithm outperforms the previous state of the art in terms of the number of instances solved, both during the refinement and when using the resulting heuristic in an A* search. In the future, we plan to evaluate the impact of our new refinement strategies when computing multiple Cartesian abstractions.

Acknowledgments

This research was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215. David Speck was supported by the German Research Foundation (DFG) as part of the EPSDAC project (MA 7790/1-1). Jendrik Seipp was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

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

Ball, T.; Podelski, A.; and Rajamani, S. K. 2001. Boolean and Cartesian Abstraction for Model Checking C Programs. In Margaria, T.; and Yi, W., eds., *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, volume 2031 of *Lecture Notes in Computer Science*, 268–283. Springer-Verlag.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-Guided Abstraction Refinement. In Emerson, E. A.; and Sistla, A. P., eds., *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, 154–169.

Drexler, D.; Seipp, J.; and Speck, D. 2021. Subset-Saturated Transition Cost Partitioning. In Goldman, R. P.; Biundo, S.; and Katz, M., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 131–139. AAAI Press.

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.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In Boddy, M.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 176–183. AAAI Press.

Katz, M.; and Domshlak, C. 2008. Optimal Additive Composition of Abstraction-based Admissible Heuristics. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 174–181. AAAI Press.

Katz, M.; Sohrabi, S.; and Udrea, O. 2020. Top-Quality Planning: Finding Practically Useful Sets of Best Plans. In Conitzer, V.; and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9900–9907. AAAI Press.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

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

Rovner, A.; Sievers, S.; and Helmert, M. 2019. Counterexample-Guided Abstraction Refinement for Pattern Selection in Optimal Classical Planning. In Lipovetzky, N.; Onaindia, E.; and Smith, D. E., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 362–367. AAAI Press.

Seipp, J. 2018. Fast Downward Scorpion. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 77–79.

Seipp, J.; and Helmert, M. 2013. Counterexample-guided Cartesian Abstraction Refinement. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 347–351. AAAI Press.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *Journal of Artificial Intelligence Research*, 62: 535–577.

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.

Seipp, J.; von Allmen, S.; and Helmert, M. 2020. Incremental Search for Counterexample-Guided Cartesian Abstraction Refinement. In Beck, J. C.; Karpas, E.; and Sohrabi, S., eds., *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, 244–248. AAAI Press.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic Top-k Planning. In Conitzer, V.; and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9967–9974. AAAI Press.

Speck, D.; and Seipp, J. 2022. Code and data for the ICAPS 2022 paper "New Refinement Strategies for Cartesian Abstractions". https://doi.org/10.5281/zenodo.6381924.

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.