Counterexample-guided Cartesian Abstraction Refinement

Jendrik Seipp and **Malte Helmert**
Universität Basel
Basel, Switzerland
{jendrik.seipp,malte.helmert}@unibas.ch

Abstract

Counterexample-guided abstraction refinement (CEGAR) is a method for incrementally computing abstractions of transition systems. We propose a CEGAR algorithm for computing abstraction heuristics for optimal classical planning. Starting from a coarse abstraction of the planning task, we iteratively compute an optimal abstract solution, check if and why it fails for the concrete planning task and refine the abstraction so that the same failure cannot occur in future iterations. A key ingredient of our approach is a novel class of abstractions for classical planning tasks that admits efficient and very fine-grained refinement. Our implementation performs tens of thousands of refinement steps in a few minutes and produces heuristics that are often significantly more accurate than pattern database heuristics of the same size.

Introduction

Counterexample-guided abstraction refinement (CEGAR) is an established technique for model checking in large systems (Clarke et al. 2000). The idea is to start from a coarse (i. e., small and inaccurate) abstraction, then iteratively improve (refine) the abstraction in only the necessary places. In model checking, this means that we search for error traces (behaviours that violate the system property we want to verify) in the abstract system, test if these error traces generalize to the actual system (called the *concrete system*), and only if not, refine the abstraction in such a way that this particular error trace is no longer an error trace of the abstraction.

Despite the similarity between model checking and planning, counterexample-guided abstraction refinement has not been thoroughly explored by the planning community. The work that comes closest to ours (Chatterjee et al. 2005) contains no experimental evaluation or indication that the proposed algorithm has been implemented. The algorithm is based on blind search, and we believe it is very unlikely to deliver competitive performance. Moreover, the paper has several critical technical errors which make the main contribution (Algorithm 1) unsound.

In model checking, CEGAR is usually used to prove the absence of an error trace. In this work, we use CEGAR to derive heuristics for optimal state-space search, and hence our CEGAR procedure does not have to completely solve the problem: abstraction refinement can be interrupted at any time to derive an admissible search heuristic.[1]

Haslum (2012) introduces an algorithm for finding lower bounds on the solution cost of a planning task by iteratively "derelaxing" its delete relaxation. Keyder, Hoffmann, and Haslum (2012) apply this idea to build a strong satisficing planning system based on the FF heuristic. Our approach is similar in spirit, but technically very different from Haslum's because it is based on homomorphic abstraction rather than delete relaxation. As a consequence, our method performs shortest-path computations in abstract state spaces represented as explicit graphs in order to find abstract solutions, while Haslum's approach exploits structural properties of delete-free planning tasks.

A key component of our approach is a new class of abstractions for classical planning, called *Cartesian abstractions*, which allow efficient and very fine-grained refinement. Cartesian abstractions are a proper generalization of the abstractions that underlie pattern database heuristics (Culberson and Schaeffer 1998; Edelkamp 2001).

Background

We consider optimal planning in the classical setting, using a SAS$^+$-like (Bäckström and Nebel 1995) finite-domain representation. Planning tasks specified in PDDL can be converted to such a representation automatically (Helmert 2009).

Definition 1. ***Planning tasks.***
*A **planning task** is a 4-tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ where:*

- *$\mathcal{V}$ is a finite set of **state variables**, each with an associated finite domain $\mathcal{D}(v)$.*
  *An **atom** is a pair $\langle v, d \rangle$ with $v \in \mathcal{V}$ and $d \in \mathcal{D}(v)$.*
  *A **partial state** is a function s defined on some subset of $\mathcal{V}$. We denote this subset by $\mathcal{V}_s$. For all $v \in \mathcal{V}_s$, we must have $s(v) \in \mathcal{D}(v)$. Where notationally convenient, we treat partial states as sets of atoms. Partial states defined on all variables are called **states**, and $\mathcal{S}(\Pi)$ is the set of all states of Π.*

[1] In the model checking community, the idea of using CEGAR to derive informative heuristics has been explored by Smaus and Hoffmann (2009), although not with a focus on optimality.

*The **update** of partial state s with partial state t, $s \oplus t$, is the partial state defined on $\mathcal{V}_s \cup \mathcal{V}_t$ which agrees with t on all $v \in \mathcal{V}_t$ and with s on all $v \in \mathcal{V}_s \setminus \mathcal{V}_t$.*

- *$\mathcal{O}$ is a finite set of **operators**. Each operator o is given by a **precondition** $pre(o)$ and **effect** $eff(o)$, which are partial states, and a **cost** $cost(o) \in \mathbb{N}_0$.*
- *s_0 is a state called the **initial state**.*
- *$s_\star$ is a partial state called the **goal**.*

The notion of transition systems is central for assigning semantics to planning tasks:

Definition 2. ***Transition systems and plans.***
*A **transition system** $\mathcal{T} = \langle S, L, T, s_0, S_\star \rangle$ consists of a finite set of **states** S, a finite set of **transition labels** L, a set of labelled **transitions** $T \subseteq S \times L \times S$, an **initial state** s_0 and a set of **goal states** $S_\star \subseteq S$. Each label $l \in L$ has an associated **cost** $cost(l)$.*

*A path from s_0 to any $s_\star \in S_\star$ following the labelled transitions T is a **plan** for $\mathcal{T}$. A plan is **optimal** if the sum of costs of the labels along the path is minimal.*

A planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ induces a transition system with states $\mathcal{S}(\Pi)$, labels $\mathcal{O}$, initial state s_0, goal states $\{s \in \mathcal{S}(\Pi) \mid s_\star \subseteq s\}$ and transitions $\{\langle s, o, s \oplus eff(o) \rangle \mid s \in \mathcal{S}(\Pi), o \in \mathcal{O}, pre(o) \subseteq s\}$. Optimal planning is the problem of finding an optimal plan in the transition system induced by a planning task, or proving that no plan exists.

Cartesian Abstractions

Abstracting a planning task means losing some distinctions between states to obtain a more "coarse-grained", and hence smaller, transition system. For this paper, it is convenient to use a definition based on equivalence relations:

Definition 3. ***Abstractions.***
Let Π be a planning task inducing the transition system $\langle S, L, T, s_0, S_\star \rangle$.

*An **abstraction relation** $\sim$ for Π is an equivalence relation on S. Its equivalence classes are called **abstract states**. We write $[s]_\sim$ for the equivalence class to which s belongs. The function mapping s to $[s]_\sim$ is called the **abstraction function**. We omit the subscript $\sim$ where clear from context.*

*The **abstract transition system** induced by $\sim$ is the transition system with states $\{[s] \mid s \in S\}$, labels L, transitions $\{\langle [s], l, [s'] \rangle \mid \langle s, l, s' \rangle \in T\}$, initial state $[s_0]$ and goal states $\{[s_\star] \mid s_\star \in S_\star\}$.*

Abstraction preserves paths in the transition system and can therefore be used to define admissible and consistent heuristics for planning. Specifically, $h^\sim(s)$ is defined as the cost of an optimal plan starting from $[s]$ in the abstract transition system. Practically useful abstractions should be efficiently computable and give rise to informative heuristics. These are conflicting objectives.

We want to construct compact and informative abstractions through an iterative refinement process. Choosing a suitable class of abstractions is critical for this. For example, *pattern databases* (Edelkamp 2001) do not allow fine-grained refinement steps, as every refinement at least doubles the number of abstract states. *Merge-and-shrink (M&S) abstractions* (Helmert, Haslum, and Hoffmann 2007) do not maintain efficiently usable representations of the preimage of an abstract state, which makes their refinement complicated and expensive.

Because of these and other shortcomings, we introduce a new class of abstractions for planning tasks that is particularly suitable for abstraction refinement. To state the definition of this class and the later example more elegantly, it is convenient to introduce a tuple notation for states. We assume that the state variables are (arbitrarily) numbered $v_1, \ldots, v_n$ and write $\langle d_1, \ldots, d_n \rangle$ to denote the state s with $s(v_i) = d_i$ for all $1 \leq i \leq n$.

Definition 4. ***Cartesian sets and Cartesian abstractions.***
*A set of states is called **Cartesian** if it is of the form $A_1 \times A_2 \times \ldots \times A_n$, where $A_i \subseteq \mathcal{D}(v_i)$ for all $1 \leq i \leq n$.*

*An abstraction is called **Cartesian** if all its abstract states are Cartesian sets.*

Figure 1, which we will discuss in detail later, shows several examples of abstract transition systems based on Cartesian abstraction. The name "Cartesian abstraction" was coined in the model-checking literature by Ball, Podelski, and Rajamani (2001) for a concept essentially equivalent to Def. 4. (Direct comparisons are difficult due to different state models.) Cartesian abstractions form a fairly general class; e. g., they include pattern databases and domain abstraction (Hernádvölgyi and Holte 2000) as special cases. Unlike these, general Cartesian abstractions can have very different levels of granularity in different parts of the abstract state space. One abstract state might correspond to a single concrete state, while another abstract state corresponds to half of the states of the task.

M&S abstractions are even more general than Cartesian abstractions because *every* abstraction function can be represented as a M&S abstraction, although not necessarily compactly. It is open whether every Cartesian abstraction has an equivalent M&S abstraction whose representation is at most polynomially larger.

Abstraction Refinement Algorithm

We now describe our abstraction refinement algorithm (Alg. 1). For a more detailed description with notes on implementation details we refer to Seipp (2012). At every time, the algorithm maintains a Cartesian abstraction $\mathcal{T}'$, which it represents as an explicit graph. Initially, $\mathcal{T}'$ is the trivial abstraction with only one abstract state. The algorithm iteratively refines the abstraction until a termination criterion is satisfied (usually a time or memory limit). At this point, $\mathcal{T}'$ can be used to derive an admissible heuristic for state-space search algorithms.

Each iteration of the refinement loop first computes an optimal solution for the current abstraction, which is returned as a *trace* τ' (i. e., as an interleaved sequence of abstract states and operators $\langle [s'_0], o_1, \ldots, [s'_{n-1}], o_n, [s_n] \rangle$ that form a minimal-cost goal path in the transition system). If no such trace exists (τ' is undefined), the abstract task is unsolvable, and hence the concrete task is also unsolvable: we are done.

Otherwise, we attempt to convert τ' into a *concrete* trace in the FINDFLAW procedure. This procedure starts from the

Algorithm 1 Refinement loop.

```
T' ← TRIVIALABSTRACTION()
while not TERMINATE() do
    τ' ← FINDOPTIMALTRACE(T')
    if τ' is undefined then
        return task is unsolvable
    φ ← FINDFLAW(τ')
    if φ is undefined (there is no flaw in τ') then
        return plan extracted from τ'
    REFINE(T', φ)
return T'
```

initial state of the concrete task and iteratively applies the next operator in τ' to construct a sequence of concrete states $s_0, \ldots, s_n$ until one of the following *flaws* is encountered:

1. Concrete state s_i does not fit the abstract state $[s'_i]$ in τ', i. e., $[s_i] \neq [s'_i]$: the concrete and abstract traces diverge. This can happen because abstract transition systems are not necessarily deterministic: the same state can have multiple outgoing arcs with the same label.
2. Operator o_i is not applicable in concrete state s_{i-1}.
3. The concrete trace has been completed, but s_n is not a goal state.

If none of these conditions occurs, we have found an optimal solution for the concrete task and can terminate. Otherwise, we proceed by refining the abstraction so that the same flaw cannot arise in future iterations. In all three cases, this is done by splitting a particular abstract state $[s']$ into two abstract states $[t']$ and $[u']$.

In the case of violated preconditions (2.), we split $[s_{i-1}]$ into $[t']$ and $[u']$ in such a way that $s_{i-1} \in [t']$ and o_i is inapplicable in all states in $[t']$. In the case of violated goals (3.), we split $[s_n]$ into $[t']$ and $[u']$ in such a way that $s_n \in [t']$ and $[t']$ contains no goal states. Finally, in the case of diverging traces (1.), we split $[s_{i-1}]$ into $[t']$ and $[u']$ in such a way that $s_{i-1} \in [t']$ and applying o_i to any state in $[t']$ cannot lead to a state in $[s'_i]$.[2] It is not hard to verify that such splits are always possible and that suitable abstract states $[t']$ and $[u']$ can be computed in time $O(k)$, where k is the number of atoms of the planning task.

Once a suitable split has been determined, we update the abstract transition system by replacing the state $[s']$ that was split with the two new abstract states $[t']$ and $[u']$ and "rewiring" the new states. Here we need to decide for each incoming and outgoing transition of $[s']$ whether a corresponding transition needs to be connected to $[t']$, to $[u']$, or both. To do this efficiently, we exploit that for arbitrary Cartesian sets X and Y and operators o, we can decide in time $O(k)$ whether a state transition from some concrete state in X to some concrete state in Y via operator o exists.

[2]Performing this split involves computing the regression of $[s'_i]$ over the operator o_i. We exploit here that regressing a Cartesian set over an operator always results in a Cartesian set. Similarly, for cases 2. and 3., we exploit that the set of states in which a given operator is applicable and the set of goal states are Cartesian.

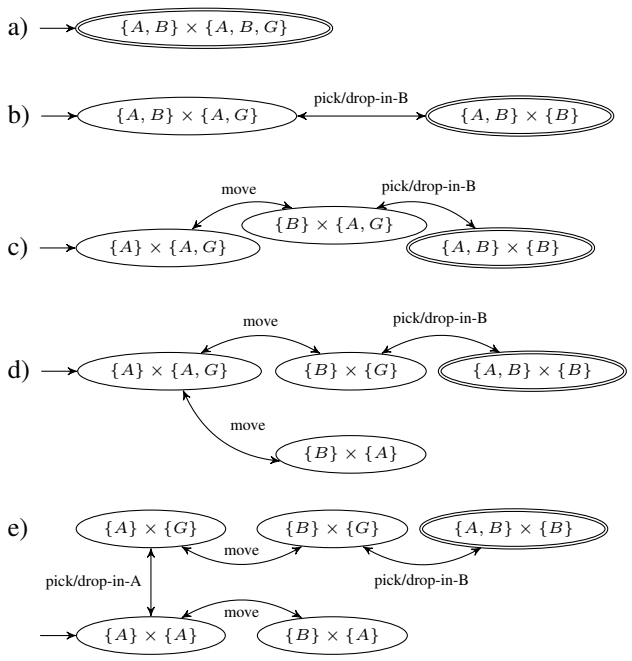

Figure 1: Refining the example abstraction. (Self-loops are omitted to avoid clutter.)

Example CEGAR Abstraction

We illustrate the creation of a CEGAR abstraction with a simple example task from the Gripper domain (McDermott 2000) consisting of a robot with a single gripper *G*, one ball and two rooms A and B. Formally the SAS$^+$ task is $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ with $\mathcal{V} = \{rob, ball\}$, $\mathcal{D}(rob) = \{A, B\}$, $\mathcal{D}(ball) = \{A, B, G\}$, $\mathcal{O} = \{$move-A-B, move-B-A, pick-in-A, pick-in-B, drop-in-A, drop-in-B$\}$, $s_0(rob) = A$, $s_0(ball) = A$, $s_\star(ball) = B$.

Figure 1a shows the initial abstraction. The empty abstract solution $\langle\rangle$ does not solve Π because s_0 does not satisfy the goal. Therefore, REFINE splits $[s_0]$ based on the goal variable, leading to the finer abstraction in Figure 1b.

The plan $\langle$drop-in-B$\rangle$ does not solve Π because two preconditions are violated in s_0: $ball = G$ and $rob = B$. We assume that REFINE performs a split based on variable *rob* (a split based on *ball* is also possible), leading to Figure 1c.

A further refinement step, splitting on *ball*, yields the system in Figure 1d with the abstract solution $\langle$ move-A-B, drop-in-B $\rangle$. The first operator is applicable in s_0 and takes us into state s_1 with $s_1(rob) = B$ and $s_1(ball) = A$, but the second abstract state $a_1 = \{B\} \times \{G\}$ of the trace does not abstract s_1: the abstract and concrete paths diverge. Regression from a_1 for move-A-B yields the intermediate state $a' = \{A\} \times \{G\}$, and hence REFINE must refine the abstract initial state $[s_0]$ in such a way that a' is separated from the concrete state s_0. The result of this refinement is shown in Figure 1e.

The solution for this abstraction is also a valid concrete solution, so we stop refining.

Coverage	h^0	h^{CEGAR}	h^{iPDB}	$h^{\text{m\&s}}_1$	$h^{\text{m\&s}}_2$
airport (50)	19	19 (13)	20	**22**	15
blocks (35)	18	18 (11)	**28**	**28**	20
depot (22)	4	4 (2)	**7**	**7**	6
driverlog (20)	7	10 (6)	**13**	12	12
elevators-08 (30)	11	16 (2)	**20**	1	12
freecell (80)	14	15 (6)	**20**	16	3
grid (5)	1	2 (1)	**3**	2	**3**
gripper (20)	7	7 (4)	7	7	**20**
logistics-00 (28)	10	14 (10)	**20**	16	**20**
logistics-98 (35)	2	3 (2)	4	4	**5**
miconic (150)	50	55 (40)	45	50	**74**
mprime (35)	19	**27** (23)	22	23	11
mystery (30)	18	**24** (15)	22	19	12
openstacks-08 (30)	**19**	18 (9)	**19**	8	**19**
openstacks (30)	7	7 (5)	7	7	7
parcprinter-08 (30)	10	11 (9)	11	15	**17**
pathways (30)	4	4 (4)	4	4	4
pegsol-08 (30)	27	27 (8)	3	2	**29**
pipesworld-nt (50)	14	15 (8)	**16**	15	8
pipesworld-t (50)	10	12 (5)	**16**	**16**	7
psr-small (50)	49	49 (46)	49	**50**	49
rovers (40)	5	6 (4)	7	6	**8**
satellite (36)	4	6 (4)	6	6	**7**
scanalyzer-08 (30)	12	12 (6)	**13**	6	12
sokoban-08 (30)	19	19 (4)	**28**	3	23
tpp (30)	5	6 (5)	6	6	**7**
transport-08 (30)	11	11 (6)	11	11	11
trucks (30)	6	7 (4)	**8**	6	**8**
woodworking-08 (30)	7	8 (7)	6	**14**	9
zenotravel (20)	8	9 (8)	9	9	**11**
Sum (1116)	397	441 (277)	**450**	391	449

Table 1: Number of solved tasks by domain. For h^{CEGAR}, tasks solved during refinement are shown in brackets.

Experiments

We implemented CEGAR abstractions in the Fast Downward system and compared them to state-of-the-art abstraction heuristics already implemented in the planner: h^{iPDB} (Sievers, Ortlieb, and Helmert 2012) and the two $h^{\text{m\&s}}$ configurations of IPC 2011 (Nissim, Hoffmann, and Helmert 2011). We applied a time limit of 30 minutes and memory limit of 2 GB and let h^{CEGAR} refine for at most 15 minutes.

Table 1 shows the number of solved instances for a number of IPC domains. While the total coverage of h^{CEGAR} is not as high as for h^{iPDB} and $h^{\text{m\&s}}_2$, we solve much more tasks than $h^{\text{m\&s}}_1$ and the h^0 (blind) baseline. We remark that h^{CEGAR} is much less optimized than the other abstraction heuristics, some of which have been polished for years. Nevertheless, h^{CEGAR} outperforms them on some domains. In a direct comparison we solve more tasks than h^{iPDB}, $h^{\text{m\&s}}_1$ and $h^{\text{m\&s}}_2$ on 5, 9 and 7 domains. While h^{CEGAR} is never the single worst performer on any domain, the other heuristics often perform even worse than h^0. Only one task is solved by h^0 but not by h^{CEGAR}, while the other heuristics fail to solve 30, 68, 40 tasks solved by h^0. These results show that h^{CEGAR} is more robust than the other approaches.

Although h^{CEGAR} typically uses far fewer abstract states, its initial plan cost estimates are often best among all approaches. On commonly solved tasks the estimates are 38%, 134% and 21% higher than those of h^{iPDB}, $h^{\text{m\&s}}_1$ and $h^{\text{m\&s}}_2$ on average. Figure 2 shows how the cost estimate for s_0 grows with the number of abstract states on an example task. The h^{CEGAR} estimates are generally higher than those of h^{iPDB} and grow much more smoothly towards the perfect estimate. This behaviour can be observed in many domains. Figure 3 shows a comparison of initial state estimates made by h^{CEGAR} and h^{iPDB} for a subset of domains.

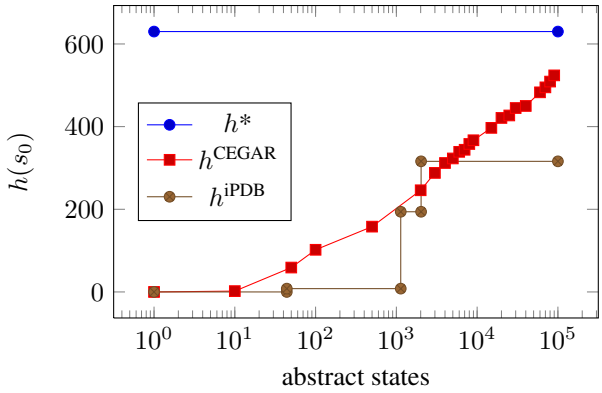

Figure 2: Initial state heuristic values for transport-08 #23.

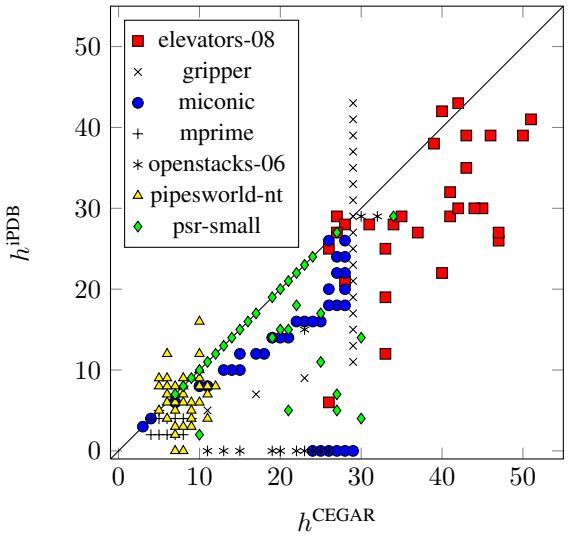

Figure 3: $h(s_0)$ for a subset of the domains where h^{CEGAR} has a better estimate of the plan cost than h^{iPDB}.

Conclusion

We introduced a CEGAR approach for classical planning and showed that it delivers promising performance. We believe that further performance improvements are possible through more space-efficient abstraction representations and speed optimizations in the refinement loop, which will enable larger abstractions to be generated in reasonable time.

All in all, we believe that Cartesian abstraction and counterexample-guided abstraction refinement are useful concepts that can contribute to the further development of strong abstraction heuristics for automated planning.

Acknowledgments

This work was supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS) and by the Swiss National Science Foundation (SNSF) as part of the project "Abstraction Heuristics for Planning and Combinatorial Search" (AHPACS).

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 *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, 268–283.

Chatterjee, K.; Henzinger, T. A.; Jhala, R.; and Majumdar, R. 2005. Counterexample-guided planning. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI 2005)*, 104–111.

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.

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

Edelkamp, S. 2001. Planning with pattern databases. In Cesta, A., and Borrajo, D., eds., *Pre-proceedings of the Sixth European Conference on Planning (ECP 2001)*, 13–24.

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 74–82. AAAI Press.

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.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

Hernádvölgyi, I. T., and Holte, R. C. 2000. Experiments with automatically created memory-based heuristics. In Choueiry, B. Y., and Walsh, T., eds., *Proceedings of the 4th International Symposium on Abstraction, Reformulation and Approximation (SARA 2000)*, volume 1864 of *Lecture Notes in Artificial Intelligence*, 281–290. Springer-Verlag.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 128–136. AAAI Press.

McDermott, D. 2000. The 1998 AI Planning Systems competition. *AI Magazine* 21(2):35–55.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1983–1990.

Seipp, J. 2012. Counterexample-guided abstraction refinement for classical planning. Master's thesis, Albert-Ludwigs-Universität Freiburg.

Sievers, S.; Ortlieb, M.; and Helmert, M. 2012. Efficient implementation of pattern database heuristics for classical planning. In Borrajo, D.; Felner, A.; Korf, R.; Likhachev, M.; Linares López, C.; Ruml, W.; and Sturtevant, N., eds., *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SOCS 2012)*, 105–111. AAAI Press.

Smaus, J.-G., and Hoffmann, J. 2009. Relaxation refinement: A new method to generate heuristic functions. In Peled, D. A., and Wooldridge, M. J., eds., *Proceedings of the 5th International Workshop on Model Checking and Artificial Intelligence (MoChArt 2008)*, 147–165.