Under-Approximation Refinement for Classical Planning

Manuel Heusner and **Martin Wehrle** and **Florian Pommerening** and **Malte Helmert**
University of Basel, Switzerland
{manuel.heusner,martin.wehrle,florian.pommerening,malte.helmert}@unibas.ch

Abstract

A general and important problem of search-based planning techniques is the state explosion problem, which is usually tackled with approaches to reduce the branching factor of the planning task. Such approaches often implicitly exploit the observation that the number of available operators is higher than the number of operators that are actually needed to find a plan. In this paper, we propose a simple, but general under-approximation refinement framework for satisficing planning that explicitly exploits this observation. Our approach iteratively searches for plans with operator *subsets*, which are *refined* if necessary by adding operators that appear to be needed. Our evaluation shows that even a straight-forward instantiation of this framework yields a competitive planner that often finds plans with small operator sets.

1 Introduction

Planning as heuristic search (Bonet and Geffner 2001) is a leading approach for domain-independent satisficing planning. A major obstacle of heuristic search based planners is the state explosion problem, i. e., the problem that the state space generally grows exponentially in the size of the planning task. To tackle this problem, various approaches have been proposed. For example, helpful actions (Hoffmann and Nebel 2001) and preferred operators (Richter and Helmert 2009) preferably apply operators that appear to be useful in the current search state. This idea has been generalized to macro operators in the YAHSP planner (Vidal 2004). All these approaches implicitly exploit the observation that the number of operators that are available in the planning task is higher than the number of operators that are actually needed to find a plan. We argue that, although obvious and intuitive, this observation deserves even more attention than it has achieved so far, as the number of available operators is often *significantly* higher than needed even in the initial state. Table 1 shows some examples for this claim in some common planning tasks. We observe that the number of operators that are needed to find a plan is sometimes (significantly) less than 1% of the available operators. Averaging over 1574 previous IPC instances solved by a greedy search with the FF heuristic (Hoffmann and Nebel 2001), 4.3% of the available operators are already sufficient to find a plan.

Task	# available	# needed
barman-sat11-p10-039	2344	72
logistics98-prob35	676	30
mystery-prob10	36738	8
nomystery-sat11-p14	4184	30
scanalyzer-sat11-strips-p18	373140	19
Geometric mean (1574 instances)		4.3%

Table 1: Number of available vs. number of needed operators (number of needed operators estimated with greedy best-first search and the FF heuristic)

In this paper, we propose a simple, yet powerful refinement framework to directly exploit this observation. Our approach iteratively searches for plans in an under-approximation of the original planning task, which is obtained by explicitly reducing the set of operators that may be applied. The operator subset is refined on-the-fly by adding some of the missing operators that appear to be needed. Our approach is parameterized with a *refinement strategy* that determines at which time which operators should be added. If at least one operator is added after finitely many steps, the resulting planning algorithm remains complete. From a high-level perspective, the overall framework is a variant of counter-example guided abstraction refinement (CEGAR) (Clarke et al. 2000). As a crucial difference to CEGAR, our under-approximation approach guarantees that no spurious plans (i. e., plans that do not correspond to valid concrete plans) are found. Our evaluation shows that even a straight-forward instantiation of this framework yields a competitive planner, which often finds plans with small operator subsets. In addition, our framework offers the potential to be instantiated in more powerful ways.

2 Preliminaries

We consider SAS$^+$ planning, where states of the world are defined with a set $\mathcal{V}$ of finite domain variables with a finite domain $Dom(v)$ for all $v \in \mathcal{V}$. A *partial state* is defined as a function s that maps a subset of variables to values of their domain. This subset where s is defined is denoted with $vars(s)$. For $v \in vars(s)$, the value of v in s is denoted with $s[v]$ (for $v \notin vars(s)$, $s[v]$ is undefined). A *state* is defined as a partial state s with $vars(s) = \mathcal{V}$.

Definition 1 (Planning task)**.** *A* planning task *is a 4-tuple* $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$*, where* $\mathcal{V}$ *is a finite set of finite-domain variables,* $\mathcal{O}$ *is a finite set of* operators*,* s_0 *is a state called the* initial state*, and* $s_\star$ *is a partial state called the* goal*.*

Operators $o \in \mathcal{O}$ have an associated partial state $pre(o)$ (the *precondition*), and an associated partial state $\mathit{eff}(o)$ (the *effect*). Furthermore, o has an associated nonnegative number $cost(o) \in \mathbb{R}_0^+$ called its *cost*. An operator is *applicable* in a state s if $s[v] = pre(o)[v]$ for all $v \in vars(pre(o))$. In this case, applying o in s yields the successor state $o(s)$, which is obtained from s by setting the values of all variables according to $\mathit{eff}(o)$, and retaining the other variables values. A *plan* π is defined as a sequence of operators $o_1 \ldots o_n$ with the property that π is applicable in sequence starting in s_0, and leads to a state that satisfies $s_\star$. The cost of π is defined as the sum of the costs of each operator in π.

We assume the reader is familiar with heuristic search algorithms and their terminology, particularly with the notion of open and closed lists (Pearl 1984). We define a state s as *fully expanded* iff all operators from $\mathcal{O}$ that are applicable in s have been applied, and the corresponding successor states have been inserted to the open list.

3 Under-Approximation Refinement

The main idea of under-approximation refinement is rather simple and can be formulated in a compact way. It is directly motivated by the fact that planning tasks often need significantly fewer operators than available to find a plan. On a high level, our approach attempts to find a plan with a (small) operator subset, and refines this subset by adding operators if necessary. In more detail, for a planning task $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$, our approach is based on the following under-approximation refinement (uar) algorithm. Let A be a best-first search planning algorithm.

- Select an *initial subset* $\mathcal{O}_0 \subseteq \mathcal{O}$ of operators. Run the planning algorithm A on $\Pi_0 := (\mathcal{V}, \mathcal{O}_0, s_0, s_\star)$ and repeat the following steps for $i = 0, 1, 2, \ldots$:
  - If a plan π is found in Π_i, then return π.
  - If the *refinement guard* triggers, then *refine* Π_i with an operator set $\mathcal{O}_{i+1}$ such that $\mathcal{O}_i \subset \mathcal{O}_{i+1} \subseteq \mathcal{O}$.
  - Re-open all closed states where at least one operator in $\mathcal{O}_{i+1} \setminus \mathcal{O}_i$ is applicable. Continue the search with $\Pi_{i+1} = (\mathcal{V}, \mathcal{O}_{i+1}, s_0, s_\star)$.

The overall algorithm has the flavor of counter-example guided abstraction refinement (CEGAR), which is a well-known approach in model checking (Clarke et al. 2000), and has recently also been applied in planning for generating abstraction heuristics (Seipp and Helmert 2013). Complementary to CEGAR, our refinement algorithm works on *under-approximations* of the original planning task Π, which allow for less behavior than Π without introducing additional behavior. As an immediate consequence, it is ensured that all found plans are guaranteed to be valid for Π, which is a crucial difference to CEGAR (where over-approximations are applied that might lead to spurious error traces and plans.)

Our uar algorithm is parametrized in several ways. In particular, there is the question *which* planning algorithm and initial operator subset to choose, *when* to refine (i. e., under which conditions the refinement guards triggers) and *how* to refine (i. e., which operators should be added to the working subset of operators). In practice, one can think of various instantiations to resolve the design choices discussed above. As a case study, we will describe a straight-forward instantiation based on relaxed plans. More sophisticated ways are discussed at the end of the paper.

3.1 An Instantiation Based on Relaxed Plans

We apply the following instantiation of the uar algorithm based on relaxed plans identified with the FF heuristic (called uar^{rp} in the following). Let $\Pi = (\mathcal{V}, \mathcal{O}, s_0, s_\star)$ be a planning task.

- Planning algorithm: Greedy best-first search, where the heuristic for the search in the intermediate under-approximations is evaluated with respect to Π (to guide the search towards a goal in the original planning task).
- Initial operator subset: The initial operator subset $\mathcal{O}_0$ is defined as the set of operators that occurs in a relaxed plan starting in s_0.
- Refinement guard: The operator subset is refined in case the open list gets empty, or possibly refined (see the next bullet) in case the search encounters a plateau or local minimum (i. e., the currently explored state has no successor state with better heuristic value).
- Refinement: The overall refinement step for operator subsets is a 4-stage process:
  1. First, select the states $s_1, \ldots, s_n$ in the closed list with lowest heuristic value l. Let $\pi_1^\#, \cdots, \pi_n^\#$ be relaxed plans that start in $s_1, \ldots, s_n$, respectively. If $\mathcal{O}' := \mathcal{O}_i \cup \{o \mid \exists j : o \text{ occurs in } \pi_j^\#\} \supset \mathcal{O}_i$, then $\mathcal{O}_{i+1} := \mathcal{O}'$, i. e., $\mathcal{O}'$ is defined as the refined subset.
  2. Otherwise, step 1 is repeated with states with heuristic values $l + 1$, $l + 2$, ..., until a refined operator subset is found, or the closed list is scanned completely.
  3. If no refined operator subset is found in step 2, and if the open list is not empty, then the search algorithm continuous its search in Π_i without refinement.
  4. If in step 3 the open list is empty, then steps 1 and 2 are repeated, with the difference that *applicable* operators (instead of operators from relaxed plans) are chosen.

Proposition 1. *The relaxed plan based instantiation of the uar algorithm is completeness preserving.*

Proof. (sketch) If the uar algorithm terminates without finding a plan, then the open list is empty, and there are no states left in closed that are not yet fully expanded. Hence, all states reachable by the underlying greedy best-first search algorithm are fully expanded, and the operators from $\mathcal{O}$ are not sufficient to find a plan in the original planning task. □

4 Evaluation

In this section, we evaluate the relaxed plan based instantiation uar^{rp} of the uar algorithm within the greedy best first search framework provided by the Fast Downward planning system (Helmert 2006).

4.1 Experimental Setup

We evaluate our approach for the h^{FF} heuristic (Hoffmann and Nebel 2001), both with and without the preferred operators enhancement. We additionally compare to the lookahead strategy (including the plan repair technique) proposed by the YAHSP2 planning system (Vidal 2011). Compared to the previously proposed version of YAHSP2, we re-implemented its lookahead strategy in Fast Downward with the (presumably more powerful) FF heuristic instead of the additive heuristic. The experiments have been performed on Intel Xeon E5-2660 CPUs (2.2 GHz) with a timeout of 30 minutes and a memory bound of 2 GB. As a benchmark set, we use the 2252 planning tasks of the 58 domains from previous IPC's in the satisficing track. For the uar^{rp} results, we additionally report the relative size of the operator subset in the last under-approximation, i. e., the geometric mean over the fraction of the available operators that has been used by the uar^{rp} algorithm to find a plan.

4.2 Experimental Results

The overall coverage results are provided in Table 2. We report the results for those domains where the coverage of the considered configurations is not equal. As a baseline, we applied the uar^{rp} algorithm with greedy best first search (GBFS) as the underlying algorithm together with h^{FF}. The baseline results (second and third column in Table 2) show the basic potential of under-approximation refinement in several aspects. Considering the size of the final operator subsets needed to find a plan (reported in parenthesis), we observe that the uar^{rp} algorithm on average only needs less than 12% of the available operators, which is already rather close to the 4.3% reported in Table 1. While there are some outliers (like PSR) with more than 50%, there are various domains where less than 5% of the operators are encountered. The reduced branching factor of the resulting under-approximations with uar^{rp} results in an overall coverage increase of 180 instances. In particular, more instances could be solved in 25 domains. On the other hand, uar^{rp} generally causes a computational overhead, and (in case the "wrong" operators are excluded from the working set) might spend too much time on searching in under-approximations. Hence, we obtain a coverage decrease in 7 domains. In particular, this happens in the Woodworking domains, where crucial operators are missing in several under-approximations, and this problem is not recognized by the relaxed plans applied in the refinement steps. To tackle this problem, more sophisticated refinement strategies are desired – we discuss a promising direction to achieve this in Section 5. On the positive side, coverage increases significantly in several domains such as the Barman and the Optical-Telegraphs domains.

The fourth and fifth column in Table 2 show the coverage overview of GBFS with h^{FF} applied with preferred operators in comparison to our under-approximation refinement framework. There are two main observations: First, the overall coverage increase obtained with the uar^{rp} algorithm is smaller than with pure GBFS, but still there is a slight improvement of 21 instances. Apart from this, we observe that the overall picture is similar to the pure GBFS setting in the sense that a coverage increase is still obtained in many domains, and the overall number of operators needed to find a plan is still low.

Finally, as our current instantiation of the uar algorithm resembles the lookaheads (LA) performed by the YAHSP planner, we re-implemented the lookahead strategy of YAHSP2 including the plan repair strategy in Fast Downward. The results when combining the LA approach with the uar^{rp} algorithm are shown in last two columns in Table 2. First, we observe that the lookahead planner with the h^{FF} heuristic yields a strong satisficing planner, which already solves 1942 out of 2252 problem instances. The overall coverage of the combined approach with uar^{rp} slightly decreases to 1935. Again, a significant role is played by the Woodworking domain, where uar^{rp} loses a large number of solved instances. Generally, we observe that the results are rather diverse: There are various domains where LA search with uar^{rp} solves more instances than LA search without uar^{rp}, and vice versa. We remark that this (empirical) diversity can also be established on a conceptual level in the sense that lookaheads and the uar^{rp} algorithm are not comparable in terms of dominance. In particular, there are planning instances Π where lookaheads do not succeed due to the structure of Π, but the uar^{rp} algorithm does.[1] Finally, we observe that the geometric mean of the overall number of operators needed to find a plan with LA and the uar^{rp} algorithm is around 6.2%, which is already very close to the 4.3% from Table 1.

Figure 1 shows the coverage as a function of time for all considered settings. Adding uar^{rp} to GBFS with h^{FF} results in a considerably higher coverage in the first second. After one minute, this setting already exceeds the coverage of h^{FF} with a 30-minute timeout. When using preferred operators, the coverage is better with uar^{rp} over the whole 30 minutes, but we see a mixed result when adding uar^{rp} to YAHSP2 lookaheads. The coverage is higher with uar^{rp} in the first 10 seconds and slightly worse after around 100 seconds.

Considering the plan quality, the average costs of the discovered plans with and without under-approximations are similar in all three configurations. In more detail, the geometric mean of the discovered plans' quality is slightly worse with uar^{rp} for GBFS with h^{FF}(factor 1.02) and for GBFS with h^{FF} and preferred operators (factor 1.01), and equal when applied with the lookahead framework.

5 Conclusions

We have presented an initial under-approximation refinement approach for classical planning, which directly exploits that the number of available operators is often significantly higher than needed to find a plan. The framework is general and can be instantiated in many different ways, which particularly offers the possibility for more advanced

[1]In a nutshell, this can happen in planning domains where the set of operators in a relaxed plan is sufficient to find a concrete plan, but in order to find this plan, the operators have to be applied multiple times.

Domain	h^{FF}			h^{FF} + preferred operators			h^{FF} + YAHSP2 lookaheads		
		uar^{rp}			uar^{rp}			uar^{rp}	
airport (50)	34	34	(20.25%)	**36**	35	(20.22%)	**28**	27	(19.48%)
barman-sat11-strips (20)	2	**20**	(15.58%)	7	**16**	(21.5%)	20	20	(1.89%)
blocks (35)	35	35	(20.98%)	35	35	(20.92%)	28	**35**	(20.74%)
depot (22)	**15**	14	(9.49%)	**18**	13	(8.61%)	**15**	12	(5.49%)
driverlog (20)	18	**19**	(14.63%)	**20**	19	(14.98%)	**16**	15	(11.54%)
elevators-sat08-strips (30)	11	**12**	(18.32%)	11	**13**	(18.74%)	12	12	(10.72%)
floortile-sat11-strips (20)	7	7	(66.55%)	7	**8**	(67.77%)	**9**	8	(68.68%)
freecell (80)	**79**	78	(1.65%)	**80**	78	(1.67%)	78	**79**	(0.67%)
grid (5)	4	**5**	(1.62%)	4	**5**	(1.62%)	5	5	(1.26%)
logistics98 (35)	30	**34**	(2.67%)	34	34	(2.65%)	30	**33**	(1.57%)
miconic-fulladl (150)	136	136	(29.39%)	137	137	(29.43%)	120	120	(22.39%)
mprime (35)	31	**34**	(0.19%)	**35**	34	(0.2%)	35	35	(0.09%)
mystery (30)	17	**18**	(0.77%)	17	**19**	(0.91%)	19	19	(0.56%)
nomystery-sat11-strips (20)	10	**15**	(9.64%)	13	**14**	(8.05%)	16	**20**	(14.94%)
openstacks-sat08-adl (30)	6	6	(76.88%)	6	6	(56.35%)	30	30	(4.99%)
openstacks-sat08-strips (30)	6	6	(74.5%)	6	6	(44.22%)	30	30	(4.99%)
openstacks-sat11-strips (20)	0	0		0	0		20	20	(1.28%)
optical-telegraphs (48)	4	**35**	(12.65%)	4	**38**	(12.65%)	48	48	(4.2%)
parcprinter-08-strips (30)	21	**23**	(33.87%)	20	**23**	(33.87%)	19	**25**	(28.84%)
parcprinter-sat11-strips (20)	5	**7**	(33.22%)	3	**7**	(33.4%)	5	**8**	(29.38%)
parking-sat11-strips (20)	20	20	(4.21%)	19	19	(3.43%)	16	**18**	(3.68%)
pathways (30)	10	**13**	(21.03%)	22	**24**	(22.45%)	30	30	(9.01%)
pathways-noneg (30)	11	**14**	(21.02%)	22	**24**	(22.01%)	30	30	(8.98%)
pegsol-08-strips (30)	30	30	(51.02%)	30	30	(51.39%)	26	26	(70.66%)
pegsol-sat11-strips (20)	20	20	(73.22%)	20	20	(70.93%)	16	16	(97.04%)
philosophers (48)	**48**	40	(55.47%)	**48**	40	(55.47%)	48	48	(17.65%)
pipesworld-notankage (50)	33	**41**	(7.51%)	42	**44**	(7.34%)	**43**	42	(4.39%)
pipesworld-tankage (50)	21	**34**	(4.24%)	34	**35**	(3.66%)	41	**43**	(0.91%)
psr-large (50)	13	13	(64.03%)	13	**14**	(66.68%)	**18**	16	(71.53%)
psr-middle (50)	38	**39**	(66.72%)	38	**39**	(70.16%)	**43**	42	(72.1%)
rovers (40)	23	**32**	(8.23%)	39	39	(7.21%)	40	40	(2.93%)
satellite (36)	27	**34**	(6.04%)	28	**34**	(6.22%)	35	35	(0.95%)
scanalyzer-08-strips (30)	28	**30**	(2.39%)	30	30	(2.67%)	26	26	(3.54%)
scanalyzer-sat11-strips (20)	18	**20**	(1.81%)	20	20	(2.04%)	16	16	(2.77%)
schedule (150)	37	**116**	(6.77%)	**150**	149	(6.84%)	146	**149**	(4.36%)
sokoban-sat08-strips (30)	**28**	27	(64.97%)	**28**	27	(63.28%)	**28**	26	(57.75%)
sokoban-sat11-strips (20)	18	18	(69.77%)	18	18	(66.9%)	**18**	17	(60.23%)
storage (30)	**18**	17	(16.65%)	**19**	16	(17.28%)	**18**	16	(18.51%)
tidybot-sat11-strips (20)	15	15	(0.64%)	15	15	(0.6%)	16	**17**	(0.82%)
tpp (30)	22	**23**	(12.41%)	30	30	(8.08%)	30	30	(3.2%)
transport-sat08-strips (30)	11	**23**	(3.82%)	20	**23**	(3.84%)	30	30	(0.63%)
transport-sat11-strips (20)	0	**4**	(1.93%)	3	**6**	(2.05%)	20	20	(0.23%)
trucks (30)	17	**18**	(4.94%)	18	18	(5.29%)	11	**14**	(4.7%)
trucks-strips (30)	17	17	(4.42%)	**18**	17	(4.71%)	11	**12**	(4.45%)
visitall-sat11-strips (20)	3	3	(100.0%)	3	3	(100.0%)	20	20	(25.95%)
woodworking-sat08-strips (30)	**27**	14	(4.92%)	**30**	14	(4.92%)	**28**	14	(4.82%)
woodworking-sat11-strips (20)	**12**	3	(1.5%)	**20**	3	(1.51%)	**17**	3	(1.51%)
Sum (2252)	1574	**1754**	(11.82%)	1808	**1829**	(11.63%)	**1942**	1935	(6.19%)

Table 2: Coverage overview: GBFS (h^{FF}, h^{FF} with preferred operators, and h^{FF} with YAHSP2 lookaheads) vs. under-approximation based on relaxed plans (uar^{rp}); the average fraction of operators needed by uar^{rp} is given in parenthesis.

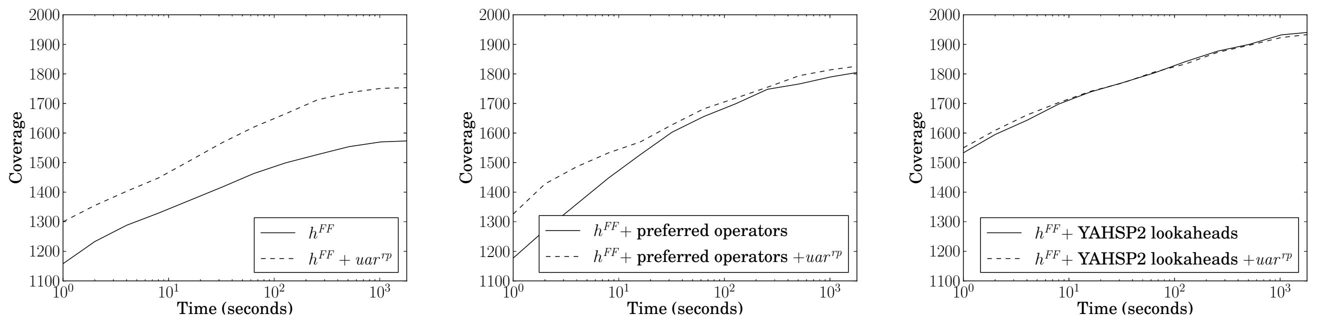

Figure 1: Coverage at different times for GBFS with h^{FF}, h^{FF} + preferred operators and h^{FF} + YAHSP2 lookaheads vs. under-approximation based on relaxed plans (uar^{rp}).

refinement strategies in the future. The experimental evaluation has served as a proof of concept and has demonstrated that a straight-forward instantiation based on relaxed plans already yields a planner that is competitive with the state of the art in satisficing planning.

Generally, we think that stronger results can be achieved with more powerful refinement strategies. Apart from the investigation of more sophisticated techniques to decide *when* a refinement step should take place, we will investigate strategies based on *linear programming* to decide *which* operators should be added. Linear programming (LP) has recently found increasing attention for planning (van den Briel, Vossen, and Kambhampati 2005; Bonet 2013; Pommerening, Röger, and Helmert 2013; Pommerening et al. 2014), particularly for the generation of powerful heuristic functions. In this setting, the heuristic value is determined by the solution of a particular LP which contains information which and how often certain operators should be applied to find a plan. Clearly, this information could directly serve as a refinement strategy as well, and will also principally generalize our approach by exploiting the information about the maximal number of operator applications in under-approximations. Generally, linear programming approaches seem to be suited well for our approach because their flexibility allows for encoding a large information spectrum. For example, the information gained from relaxed plans $\pi^{\#}$ can be encoded such that solutions are forced to contain at least the operators from $\pi^{\#}$, which again will generalize our current approach.

Acknowledgments

This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Safe Pruning in Optimal State-Space Search (SPOSSS)".

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *AIJ* 129(1):5–33.

Bonet, B. 2013. An admissible heuristic for SAS$^+$ planning obtained from the state equation. In *Proc. IJCAI 2013*, 2268–2274.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-guided abstraction refinement. In *Proc. CAV 2000*, 154–169.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

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

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *Proc. ICAPS 2014*.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In *Proc. IJCAI 2013*, 2357–2364.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proc. ICAPS 2009*, 273–280.

Seipp, J., and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In *Proc. ICAPS 2013*, 347–351.

van den Briel, M.; Vossen, T.; and Kambhampati, S. 2005. Reviving integer programming approaches for AI planning: A branch-and-cut framework. In *Proc. ICAPS 2005*, 310–319.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proc. ICAPS 2004*, 150–159.

Vidal, V. 2011. YAHSP2: Keep it simple, stupid. In *IPC 2011 planner abstracts*, 83–90.