Making Hill-Climbing Great Again through Online Relaxation Refinement and Novelty Pruning

Maximilian Fickert
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
fickert@cs.uni-saarland.de

Abstract

Delete relaxation is one of the most successful approaches to classical planning as heuristic search. The precision of these heuristics can be improved by taking *some* delete information into account, in particular through atomic *conjunctions* in the $h^{C\text{FF}}$ heuristic. It has recently been shown that this heuristic is especially effective when these conjunctions are learned online in a hill-climbing search algorithm. In this work, we devise a natural extension to this approach using *novelty pruning*, a recently-developed technique that prunes states based on whether they contain facts not seen before in the search. We evaluate our extension on the IPC benchmarks, where it beats LAMA, Mercury, and Dual-BFWS on many domains.

Introduction

In classical planning, a task forms a deterministic transition system where states are represented by a set of propositional facts. Transitions are described by actions that have sets of facts as preconditions, and applying an action deletes and adds facts. The goal is to find a sequence of actions leading from a given initial state to a goal state. A prominent approach to solve classical planning tasks is heuristic search (e.g. (Hoffmann and Nebel 2001; Richter and Westphal 2010; Domshlak, Hoffmann, and Katz 2015)).

In satisficing planning, heuristics based on the delete relaxation were part of most state-of-the-art planners for almost two decades. However, the delete relaxation often ignores critical features of the planning task, e.g. fuel consumption. These pitfalls can be diminished by "un-relaxing" part of the problem. One such method is to respect certain combinations of facts in the relaxed plans, e.g. being in a specific location while still having a certain amount of fuel. The $h^{C\text{FF}}$ heuristic implements this by treating a set of conjunctions C as atomic (Fickert, Hoffmann, and Steinmetz 2016). Choosing C correctly is critical for the performance of the heuristic, since, while the accuracy increases with larger C, so does the computational complexity.

Fickert and Hoffmann (2017a) have recently shown that the $h^{C\text{FF}}$ heuristic is particularly effective when the conjunctions are generated online. They employ a variant of enforced hill-climbing (Hoffmann and Nebel 2001), called Refinement-HC, to detect when the search is stuck in a local minimum. When this happens, the heuristic is refined until the local minimum is removed from the search space surface. In standard enforced hill-climbing, the search progresses through iterations of breadth-first search until a new best state is found, then continuing from there. In Refinement-HC, the depth of the breadth-first search explorations is bounded. Whenever the search fails to find a state with lower h-value within that bound, conjunctions are added to C. Thereby, local minima are escaped not by search, but by heuristic refinement.

In this work, we extend the Refinement-HC approach with *novelty pruning* (Lipovetzky and Geffner 2012). This pruning technique ignores states that do not contain at least one *novel* fact (not contained in any of the states generated so far). Despite its simplicity, novelty pruning performs very well on many standard planning benchmarks (e.g. (Lipovetzky and Geffner 2017a; Katz et al. 2017; Lipovetzky and Geffner 2017b)).

In order to adapt EHC for online-refinement, the main modification is defining a criterion when to refine the heuristic. In Refinement-HC, this is done by bounding the depth of the breadth-first search iterations. Here, we bound the breadth-first search explorations using incomplete novelty pruning instead. This makes the local explorations more flexible, as it allows promising regions (with novel facts) to be explored in more depth. We evaluate our algorithm on the IPC benchmarks, where it consistently improves the performance of Refinement-HC and outperforms state-of-the-art planners on many domains.

Background

We start by introducing the basic planning definitions, before briefly summarizing novelty pruning, as well as $h^{C\text{FF}}$ and how it is used in Refinement-HC.

Planning Framework

We use the STRIPS formulation of planning tasks (Fikes and Nilsson 1971), where each state is a set of boolean facts. A STRIPS *task* is defined as a 4-tuple $\Pi = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{F}$ is a set of *facts*, $\mathcal{A}$ are the *actions*, $\mathcal{I}$ is the *initial state*, and $\mathcal{G}$ a set of *goal facts*. Each action $a = \langle pre_a, add_a, del_a \rangle$ is a tuple of its *preconditions*, *add effects*, and *delete effects*, each being a subset of $\mathcal{F}$, with $pre_a \cap del_a = \emptyset$. We assume unit action costs.

A *state* $s \subseteq \mathcal{F}$ is a set of facts. An action $a \in \mathcal{A}$ is *applicable* in s if $pre_a \subseteq s$, and applying a in s results in the state $appl(s, a) = (s \setminus del_a) \cup add_a$. A *plan* for s is an applicable action sequence leading from s to a goal state. A plan for a task Π is a plan for its initial state $\mathcal{I}$. A plan is called *optimal* if it has minimal length among all plans.

We will need the concept of regression, which yields the set of facts required to achieve a subgoal through a given action. A set of facts g is *regressable* over an action a if $add_a \cap g \neq \emptyset$ and $del_a \cap g = \emptyset$. In this case, the *regression* of g over a is defined as $R(g, a) = (g \setminus add_a) \cup pre_a$, otherwise we write $R(g, a) = \perp$.

The set of all states is denoted by $\mathcal{S}$. A *heuristic function* (short *heuristic*) $h : \mathcal{S} \mapsto \mathbb{N}_0 \cup \{\infty\}$ is a function that assigns an estimated goal distance to each state, or ∞ to indicate that no plan exists for a state (such a state is called *dead end*). For simplicity, we assume that $h(s) = 0$ iff $s \supseteq \mathcal{G}$. The *perfect heuristic* h^* assigns each state s the length of an optimal plan for s, or ∞ no plan exists for s.

Given a solvable task Π, a search algorithm is *complete* if it always finds a solution for Π in finite time.

Novelty Pruning

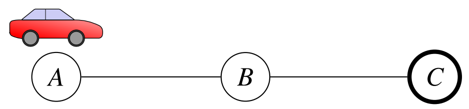
A set of facts is called *novel* if it was not contained in any previously generated state. The *novelty* of a state s is the size of the smallest novel tuple of facts in s. The concept of *novelty* has been exploited in different forms, in particular as a pruning function in Iterated Width Search (IW) (Lipovetzky and Geffner 2012). A single iteration IW(k) is a plain breadth-first search in which all states with novelty greater than k are pruned. IW(k) expands at most $|\mathcal{F}|^k$ states, making it incomplete (unless $k = |\mathcal{F}|$, when only duplicate states are pruned). IW performs successive iterations of IW(k) with increasing k. Lipovetzky and Geffner have shown that many domains (independent of the selected instances) are solvable with low and bounded k when restricting the goal to a single fact.

Intuitively, the success of these width-based methods can be explained by the observation that actions in (optimal) plans often achieve at least one fact (or small tuple of facts) that was not true in any preceding state on the solution path.

Delete Relaxation and $h^{C\mathrm{FF}}$

The *delete relaxation* is a simplified task where the delete effects of all actions are assumed to be empty. The *perfect delete relaxation heuristic* h^+ estimates the remaining goal distance of a state s by the distance of an optimal delete relaxed plan for s (or ∞ if no delete relaxed plan for s exists). Computing h^+ is not possible in polynomial time, which makes it unsuitable to use as a heuristic in search. In practice, h^{FF} is used instead, which does not require the delete relaxed plans to be optimal.

Example 1 *Consider the task illustrated below. The car has to move from A to C. The car can only hold one unit of fuel, which each drive action consumes, but can be refueled at any location. Formally, there are facts at(x) for the position of the car and fuel to indicate if the car has fuel. Initially the car is at location A and holds fuel.*



An optimal plan is $\langle$*drive(A, B), refuel, drive(B, C)*$\rangle$. *A fully delete relaxed plan can ignore the fuel consumption:* $\langle$*drive(A, B), drive(B, C)*$\rangle$.

Delete relaxation heuristics can be made more accurate by taking *some* delete information into account. The $h^{C\mathrm{FF}}$ heuristic accomplishes this by treating a set of conjunctions C of facts as atomic. Achieving a conjunction $c \in C$ means achieving the individual facts represented by c *simultaneously*. Whenever a conjunction is a subset of the preconditions of an action, the conjunction of these facts must be achieved instead of the facts individually. Throughout this paper we implicitly assume C to contain at least all singleton conjunctions $c = \{f\}$ for all facts $f \in \mathcal{F}$.

The partially relaxed plans for $h^{C\mathrm{FF}}$ (C-relaxed plans) consist of pairs of actions and sets of supported conjunctions (a, C') called *action occurrences*. The set of supported conjunctions C' indicates which conjunctions the action is achieving. An action a can only achieve a conjunction c if $R(c, a) \neq \perp$. The action occurrence has additional preconditions for the parts of the supported conjunctions that are not achieved by the action itself, so the overall preconditions of an action occurrence are $pre_{(a,C')} = (\bigcup_{c \in C'} R(c, a))^C$, where "$X^C$" is a shorthand for $\{c \in C \mid c \subseteq X\}$.

Example 2 *Assume C consists of the conjunction $c = \{$at(B), fuel$\}$ (and all singletons). Since $c \subseteq pre_{\textit{drive(B, C)}}$, partially relaxed plans must achieve c instead of at(B) and fuel individually before drive(B, C) can be applied.*

The only actions that could potentially make c true are the actions achieving part of it (i.e. driving to B and the refuel action). However, the drive actions delete the fuel part of c, so c can only be achieved using refuel. With the refuel action, at(B) must already be true to achieve c. The action occurrence $(\textit{refuel}, \{c\})$ has preconditions $R(c, \textit{refuel}) = \{\textit{at(B)}\}^C = \{\{\textit{at(B)}\}\}$. Thus, a C-relaxed plan is $\langle(\textit{drive(A, B)}, \{\{\textit{at(B)}\}\})$, $(\textit{refuel}, \{\{\textit{at(B)}, \textit{fuel}\}\})$, $(\textit{drive(B, C)}, \{\{\textit{at(C)}\}\})\rangle$, which is also a real plan.

The set of conjunctions C is typically generated by an iterative refinement procedure based on counter-example guided abstraction refinement. The algorithm identifies conflicts in the current relaxed plan, and adds conjunctions that prevent these conflicts from occurring in the next computed plan (Haslum 2012; Keyder, Hoffmann, and Haslum 2012; 2014; Fickert and Hoffmann 2017b). This procedure can be repeated until some condition is met, e.g. a time or memory bound. The $h^{C\mathrm{FF}}$ heuristic converges in the sense that with sufficiently large C, all C-relaxed plans are real plans.

Refinement-HC

The $h^{C\mathrm{FF}}$ heuristic works best when the conjunctions are generated online, in particular in an adaptation of enforced hill-climbing (Fickert and Hoffmann 2017a).

Standard enforced hill-climbing (EHC) performs iterations of breadth-first search (BrFS) until a state with lower heuristic value is found, then starting the next BrFS phase from that state until a goal is reached (Algorithm 1).

Algorithm 1: Enforced Hill Climbing

```
$s_{best} := I$
while $h(s_{best}) \neq 0$ do
    Run BrFS from $s_{best}$ until a state $s$ with
      $h(s) < h(s_{best})$ is found.
    if no such state exists then
        return FAIL
    $s_{best} := s$
return SOLVED
```

Refinement-HC extends EHC in two ways. First, it introduces a criterion upon which the heuristic is refined (the "refinement trigger"). This is achieved by placing a depth bound d on the BrFS exploration, which we write as BrFS$[d]$. Whenever the BrFS phase fails to find a state with better heuristic value than s_{best}, refinement is triggered (this essentially replaces the *FAIL* case). Second, Refinement-HC restarts from the initial state (without resetting the heuristic), when it detects that s_{best} is a dead end. Together with the converging heuristic function refinement, this makes the search algorithm complete.

In principle, BrFS$[d]$ could be exchanged by any other local exploration strategy. In this work, we show that simply replacing the depth bound by novelty pruning improves the overall performance of the search algorithm significantly, as it explores the local search space in a more targeted manner.

Refinement-HC with Novelty Pruning

In Refinement-HC, the local exploration is the most critical part of the overall search algorithm. The performance highly depends on the depth bound d for BrFS$[d]$, as it controls how much refinement is done. Selecting a smaller value for d makes the local explorations more restricted and triggers the refinement earlier, while larger values for d can allow the search to escape the local minimum through brute-force search instead of refining the heuristic. However, a simple BrFS is not always a good choice, as it ignores the structure of the local search space.

We can make the local exploration more directed by limiting BrFS through novelty pruning instead of using a depth bound. The refinement is triggered when the local exploration runs out of states passing the novelty test. This accomplishes the same goal of restricting the BrFS exploration, but it allows certain branches (those with states that pass the novelty test) to be explored in more depth, beyond the bound of BrFS$[d]$. Furthermore, it avoids states that do not contain any new facts (and thus are less likely to have shorter C-relaxed plans to the goal), which can reduce the search effort of the local exploration.

In our adapted Refinement-HC, we replace BrFS$[d]$ by IW(k), i.e. BrFS with novelty pruning (Algorithm 2). The novelty is only tracked within a single BrFS exploration starting from s_{best}, not across the overall search. This is because (a), we want to use novelty pruning to bound the BrFS exploration, not as a pruning method in the overall search, and (b), to retain the completeness property of Refinement-HC. Regarding (b), if the novelty takes into account all states explored in the overall search, s_{best} could wrongly become a dead end because all successors are pruned, thereby losing completeness. Using IW(k) in the described way instead of BrFS$[d]$ as the local exploration strategy only changes the way the local search space around s_{best} is explored, and retains the completeness of Refinement-HC which is ensured by the convergence of $h^{C\text{FF}}$.

Algorithm 2: Refinement-HC with Novelty Pruning (simplified)

```
$s_{best} := I$
while $h(s_{best}) \neq 0$ do
    Run IW($k$) from $s_{best}$ until a state $s$ with
      $h(s) < h(s_{best})$ is found.
    if no such state exists then
        Refine $h$ in $s_{best}$.
        continue
    $s_{best} := s$
return SOLVED
```

Novelty Pruning with Conjunctions

In IW(k), novelty pruning is used to prune states with novelty greater than k, i.e. states not containing a novel k-tuple of facts. We can generalize this from k-tuples to an arbitrary set of conjunctions:

Definition 1 (IW(C)) *For a set of conjunctions C, we define IW(C) as a breadth-first search that prunes all states s that do not contain a novel conjunction $c \in C, c \subseteq s$.*

This is not an entirely new idea (it is mentioned e.g. in the conclusion of Katz et. al's (2017) work). However, it has not been implemented in practice yet. The main reason is that it is not clear how a suitable set of conjunctions can be generated for novelty pruning. Since in our setting there is a set of conjunctions readily available, i.e. the set of conjunctions used by the $h^{C\text{FF}}$ heuristic, we implemented a variant of our algorithm where these conjunctions are used for novelty pruning as well. Whenever a conjunction is added to $h^{C\text{FF}}$, it is also added for the novelty pruning in IW(C).

Sharing the set of conjunctions with $h^{C\text{FF}}$ also has a synergistic side-effect in Refinement-HC. The $h^{C\text{FF}}$ heuristic becomes more expensive to evaluate with each added conjunction, so refinement should be used carefully. On the other hand, IW(C) is less restrictive with each added conjunction. Thus, as Refinement-HC progresses, refinement will be triggered less frequently with larger C (since the novelty pruning is less aggressive), which reduces further overhead for $h^{C\text{FF}}$.

Experiments

We implemented our techniques in Fast Downward (Helmert 2006). The experiments were run on a cluster of Intel Xenon E5-2650 v3 processors with a clock rate of 2.3 GHz. The time and memory limits were set to 30 minutes and 4 GB respectively. We ran our experiments on all domains from the satisficing tracks of past IPCs.

All our Refinement-HC configurations use the best performing parameter settings from Fickert and Hoffmann's (2017a) work, in particular $d = 3$ for BrFS[d]. Since the algorithm uses random tie breaking for both the heuristic and the successor generation, we averaged the results over 10 different random seeds. We compare our approach to Refinement-HC and the state of the art, represented by LAMA (Richter and Westphal 2010), Mercury (Domshlak, Hoffmann, and Katz 2015), and the novelty-based Dual-BFWS planner (Lipovetzky and Geffner 2017a).

Coverage Results

Table 1 shows the coverage results on the IPC benchmarks. Both the IW(1) and IW(C) configurations greatly improve over the Refinement-HC configuration using BrFS[3] ($+37.8$ respectively $+47.7$ total coverage). The increase in coverage can be observed across almost all domains, most significantly in Parking ($+12.1$ with IW(C)), Sokoban ($+9.5$), and Barman ($+4.6$). The only domains with losses in coverage are Transport (-3.6) and Nomystery (-1.5). The Refinement-HC configuration using IW(2) does not perform well, because the refinement is triggered too late as discussed in the next subsection.

Using IW(C) as the local exploration works best overall, and is consistently the best option across almost all domains. It beats both LAMA and Mercury overall ($+41.8$ respectively $+11.8$), and only loses very slightly to Dual-BFWS (-2.2). In 12 domains, it is strictly better than both LAMA and Mercury, and worse than either in 9 domains. Comparing directly to Dual-BFWS, Refinement-HC with IW(C) is better in 16 domains and worse in 13. The domains with the biggest advantage are those where BrFS with bounded depth already works very well, e.g. Floortile (+32/32/30 compared to LAMA/Mercury/Dual-BFWS) or Maintenance (+17/10/9). Conversely, in domains where the other planners are much stronger, Refinement-HC does not perform well irrespective of the applied local exploration strategy. Examples are Sokoban ($-32/26/27$), where local search algorithms perform poorly due to many dead ends, or VisitAll (-20.3), where delete relaxation heuristics get lost in large plateaus which cannot be efficiently resolved through conjunctions in $h^{C\text{FF}}$.

Below the coverage results, Table 1 shows the run times of the planners on commonly solved instances. On average, Refinement-HC with IW(C) is the fastest planner. It is roughly 1.3x faster than LAMA, 1.4x faster than Mercury, and more than 3x faster than Dual-BFWS. Due to the randomized nature of Refinement-HC, the search time can vary slightly between different random seeds (the relative standard deviation is 31%). The slow run times of Dual-BFWS are because it runs incomplete searches first, and only if those do not succeed a complete search algorithm is started. While this strategy can sometimes find solutions very fast, it also adds overhead on instances where the incomplete phase fails to find a solution.

Search	Refinement-HC				LAMA	Merc.	Dual-BFWS
Local Expl.	IW(1)	IW(2)	IW(C)	BrFS[3]			
Airport 50	46.5	33.9	46.5	43.4	32	32	**47**
Barman 40	36.8	5.2	37.7	33.1	39	**40**	**40**
Cavediving 20	7.0	7.0	**7.1**	**7.1**	7	7	7
Childsnack 20	5.6	5.6	5.2	3.3	5	0	**9**
CityCar 20	10.7	4.4	10.5	9.6	3	5	**20**
Depot 22	**22.0**	**22.0**	**22.0**	**22.0**	20	21	21
DriverLog 20	19.7	17.8	**20.0**	19.7	**20**	**20**	**20**
Elevators 50	**50.0**	48.9	**50.0**	**50.0**	**50**	**50**	**50**
Floortile 40	**40.0**	38.7	**40.0**	**40.0**	8	8	10
Freecell 80	74.4	77.3	76.6	73.0	79	**80**	**80**
GED 20	**20.0**	13.0	**20.0**	**20.0**	**20**	**20**	15
Hiking 20	**20.0**	19.7	**20.0**	19.9	18	**20**	8
Logistics 63	**63.0**	57.5	**63.0**	59.5	**63**	**63**	62
Mainten. 20	**17.0**	10.9	**17.0**	16.9	0	7	8
Nomystery 20	9.5	11.4	9.7	11.2	10	14	**16**
Openst. 100	**100.0**	96.0	**100.0**	**100.0**	**100**	**100**	98
Parcpr. 50	**50.0**	**50.0**	**50.0**	**50.0**	49	**50**	43
Parking 40	39.9	25.8	**40.0**	27.9	**40**	**40**	**40**
Pathways 30	**30.0**	**30.0**	**30.0**	29.9	23	**30**	29
Pegsol 50	46.7	49.6	49.8	48.4	**50**	**50**	**50**
Pipes-NT 50	45.3	42.8	45.5	44.7	43	44	**48**
Pipes-T 50	43.4	42.9	**44.1**	42.7	42	42	34
Rovers 40	**40.0**	34.4	**40.0**	**40.0**	**40**	**40**	**40**
Satellite 36	**36.0**	34.7	**36.0**	35.8	**36**	**36**	30
Scanalyzer 50	**50.0**	48.8	**50.0**	**50.0**	**50**	**50**	48
Sokoban 50	14.8	15.4	16.0	6.5	**48**	42	43
Storage 30	**29.1**	22.4	28.2	27.3	19	19	28
Tetris 20	15.8	1.0	15.5	11.9	12	**19**	17
Thoughtful 20	19.9	19.3	**20.0**	19.7	16	13	16
Tidybot 20	16.0	17.9	17.7	15.1	17	14	**18**
TPP 30	**30.0**	28.0	**30.0**	**30.0**	**30**	**30**	**30**
Transport 70	53.2	32.7	53.3	56.9	61	**70**	**70**
Trucks 30	14.9	15.1	15.7	15.0	15	**19**	14
VisitAll 40	19.7	4.8	19.7	18.6	**40**	**40**	**40**
Others 414	**414.0**	**414.0**	**414.0**	**414.0**	**414**	**414**	**414**
Sum 1725	1550.9	1398.9	1560.8	1513.1	1519	1549	**1563**
Time (s)	0.27	1.19	**0.25**	0.33	0.33	0.36	0.78
Exp. until R.	14.0	30.8	15.8	17.7	–	–	–
Refinement	21.1	10.6	19.0	36.0	–	–	–

Table 1: Coverage. Comparison of different local search strategies in Refinement-HC with LAMA, Mercury, and Dual-BFWS. The columns on the left show Refinement-HC with different local search configurations: three configurations with novelty-based exploration, and the default configuration with depth-bounded BrFS. Domains solved by all configurations are grouped to "Others". The row below the overall coverage shows the geometric mean of the planner run times on commonly solved instances in seconds. The last two rows show refinement statistics.

Refinement Behavior

We now look at the refinement behavior of the different local exploration strategies. The last two rows of Table 1 show the number of expansions in a local exploration preceding a refinement ("Exp. until R.") and the number of times refinement was triggered during search ("Refinement"), both as the geometric mean over all commonly solved instances where refinement was triggered at least once. Both IW(1) and IW(C) expand a similar number of states as BrFS[3] in local exploration phases where the heuristic is refined afterwards (c.f. Exp. until R.). However, with IW(1) and IW(C), refinement is triggered much less frequently overall: 21.1 respectively 19.0 times compared to 36.0 times for BrFS[3]. This shows that a state with lower heuristic value can be found more often in the local exploration phase with the novelty-based exploration strategies.

With IW(2), refinement is triggered too late and the heuristic does not become accurate enough. Additionally, with IW(2), on average 33% of the search time is spent evaluating the novelty compared to around 2% for IW(1) and IW(C).

Conclusion

We have introduced a natural extension for Refinement-HC using a novelty-based local exploration. We have shown that this can significantly increase the empirical performance, especially when using the conjunction set from $h^{C\text{FF}}$ in the novelty pruning as well. While our hill-climbing-based approach beats the state of the art in many domains, it cannot completely overcome the limitations of local search in domains with deep dead ends (Sokoban) or domains where the local minima cannot be effectively removed through $h^{C\text{FF}}$ refinement (VisitAll).

For future work, it may be interesting to explore whether novelty pruning could help with online-refinement of $h^{C\text{FF}}$ in GBFS as well, which has lagged behind Refinement-HC in terms of performance so far.

Acknowledgments

This work was partially supported by the German Research Foundation (DFG), under grant HO 2169/5-1, "Critically Constrained Planning via Partial Delete Relaxation". We thank the anonymous reviewers, whose comments helped significantly to improve this paper.

References

Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds. 2012. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence* 221:73–114.

Fickert, M., and Hoffmann, J. 2017a. Complete local search: Boosting hill-climbing through online heuristic-function refinement. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)* (2017).

Fickert, M., and Hoffmann, J. 2017b. Ranking conjunctions for partial delete relaxation heuristics in planning. In Fukunaga, A., and Kishimoto, A., eds., *Proceedings of the 10th Annual Symposium on Combinatorial Search (SOCS'17)*. AAAI Press.

Fickert, M.; Hoffmann, J.; and Steinmetz, M. 2016. Combining the delete relaxation with critical-path heuristics: A direct characterization. *Journal of Artificial Intelligence Research* 56(1):269–327.

Fikes, R. E., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. In Bonet et al. (2012), 74–82.

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

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

2017. *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*, AAAI Press.

Katz, M.; Lipovetzky, N.; Moshkovich, D.; and Tuisov, A. 2017. Adapting novelty to classical planning as heuristic search. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)* (2017), 172–180.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In Bonet et al. (2012), 128–136.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2014. Improving delete relaxation heuristics through explicitly represented conjunctions. *Journal of Artificial Intelligence Research* 50:487–533.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In Raedt, L. D., ed., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, 540–545. Montpellier, France: IOS Press.

Lipovetzky, N., and Geffner, H. 2017a. Best-first width search: Exploration and exploitation in classical planning. In Singh, S., and Markovitch, S., eds., *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, 3590–3596. AAAI Press.

Lipovetzky, N., and Geffner, H. 2017b. A polynomial planning algorithm that beats LAMA and FF. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)* (2017), 195–199.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.