Best-First Width Search for Lifted Classical Planning

Augusto B. Corrêa[1] and **Jendrik Seipp**[2]

[1]University of Basel, Switzerland
[2]Linköping University, Sweden
augusto.blaascorrea@unibas.ch, jendrik.seipp@liu.se

Abstract

Lifted planners are useful to solve tasks that are too hard to ground. Still, computing informative lifted heuristics is difficult: directly adapting ground heuristics to the lifted setting is often too expensive, and extracting heuristics from the lifted representation can be uninformative. A natural alternative for lifted planners is to use width-based search. These algorithms are among the strongest for ground planning, even the variants that do not access the action model. In this work, we adapt best-first width search to the lifted setting and show that this yields state-of-the-art performance for hard-to-ground planning tasks.

Introduction

Classical planning tasks are usually defined in a *lifted* (i.e., first-order) representation via the planning domain definition language (PDDL) (McDermott 2000; Haslum et al. 2019). Still, almost all state-of-the-art heuristic search planners *ground* the task to a propositional representation before starting the search (e.g., Bonet and Geffner 2001; Hoffmann and Nebel 2001; Helmert 2006; Lipovetzky and Geffner 2012). While this approach has certainly proven successful, there are planning tasks where grounding tasks is intractable. For such tasks, *lifted planners* (Ridder and Fox 2014; Corrêa et al. 2020; Lauer et al. 2021) can be particularly useful. These planners use the PDDL encoding directly without grounding the task prior to the search.

A challenge faced by lifted planners is how to obtain strong search guidance. So far, two approaches for this challenge have been presented. First, we can compute the same heuristics used for ground planning, but this can be more expensive than the same computation on the ground task (Corrêa et al. 2021). Second, we can simplify the lifted representation of the task before computing a heuristic with the price of reducing the informativeness of the resulting heuristic (Ridder and Fox 2014; Lauer et al. 2021).

In this work, we present a third approach: using *best-first width search* (BFWS) (Lipovetzky and Geffner 2017). A width-based search evaluates states based on *novelty* criteria (Lipovetzky and Geffner 2012), which are usually very fast to evaluate. As a result, width-based search planners have been very successful in the International Planning Competition (IPC) 2018 satisficing and agile tracks (Francès et al. 2018). Furthermore, these methods also excel in simulation settings, where the planner does not have access to an action model of the task (Francès et al. 2017). This is similar to the setting of lifted planning, where obtaining the ground actions is too expensive. This correspondence is a motivator for our work, since it suggests that there is a synergy between lifted planning and width-based search.

To evaluate this hypothesis, we analyze how width-based search algorithms perform in *hard-to-ground domains* (Lauer et al. 2021). We show that the ground planner versions do not scale to these tasks because either the grounding step is intractable, or the ground representation is simply too large and adds too much overhead to the search. Implementing best-first width search algorithms in a lifted planner requires keeping track of the reachable atoms as they are discovered. By making this tracking efficient, we obtain a lifted planner that outperforms state-of-the-art lifted planners in hard-to-ground domains and is even competitive with its ground counterpart in many standard IPC domains. We also present a method for combining BFWS with other heuristics using greedy best-first search with multiple open-list queues. Doing so, we add an exploitative aspect to the exploration-focused BFWS, which leads to solving more tasks in several planning domains.

Classical Planning

A *lifted STRIPS planning task* (Fikes and Nilsson 1971) is a tuple $\Pi = \langle \mathcal{P}, \mathcal{C}, \mathcal{A}, s_0, \gamma \rangle$, where $\mathcal{P}$ is a finite set of *predicates*, $\mathcal{C}$ is a finite set of *constants*, $\mathcal{A}$ is a finite set of *action schemas*, s_0 is the *initial state*, and the set of ground atoms γ is the *goal*.

An action schema $A \in \mathcal{A}$ consists of three different sets of atoms: *pre*(A), its *precondition*; *add*(A), the *add list*; and *del*(A), the *delete list*. All atoms occuring in A are from $\mathcal{P}$. We write $A(\mathbf{x})$ to denote that $\mathbf{x}$ is the set of variables occurring in any atom of *pre*$(A) \cup$ *add*$(A) \cup$ *del*(A). We *ground* an action schema $A(\mathbf{x})$ by replacing all variables $\mathbf{x}$ in *pre*(A), *add*(A), and *del*(A) with constants from $\mathcal{C}$ according to a substitution function. An action schema $A(\mathbf{x})$ with $\mathbf{x} = \emptyset$ is a *ground action*.

A *state* s is a set of ground atoms (i.e., without variables) that are true in this particular situation. The *initial state* s_0

contains the atoms that are initially true. The set γ represents the set of atoms that must hold simultaneously to solve the task. All states s_* where $\gamma \subseteq s_*$ are called *goal states*.

The precondition *pre*(A) of a ground action A is *satisfied* in a state s iff *pre*$(A) \subseteq s$. This means that action A is *applicable* in s. The *successor state* s' is defined as $s' = (s \setminus \mathit{del}(A)) \cup \mathit{add}(A)$. A sequence of actions $A_1, \ldots, A_n$ is applicable in a given state s_0 if there are states $s_1, \ldots, s_n$ such that A_i is applicable in s_{i-1} and the successor of s_{i-1} through A_i is state s_i. An atom P is *reachable* if there is an applicable sequence of actions ending in a state where P is true. An applicable sequence of actions from s_0 to some goal state is called a *plan*.

Classical planning tasks are usually modeled in PDDL (McDermott 2000), which covers the STRIPS formalism. However, most planners use *ground* planning tasks (e.g., Hoffmann and Nebel 2001; Helmert 2006; Francès et al. 2018). We say that a planning task is ground if all its actions are ground. To obtain a ground representation, planners apply a preprocessing step that instantiates action schemas with the constants of the task (e.g., Helmert 2009).

For many larger tasks, this grounding process becomes a bottleneck: ground planners either cannot represent the task internally or consume too much memory and time to ground them. We call such tasks *hard-to-ground* tasks (Corrêa et al. 2020). The alternative to grounding is to use planners that rely exclusively on the PDDL representation and avoid grounding altogether. Recent works show that such *lifted* planners outperform ground planners in hard-to-ground tasks (Corrêa et al. 2021; Lauer et al. 2021).

Best-First Width Search

Best-first width search (Lipovetzky and Geffner 2017) is a search algorithm that uses *novelty measures* (Lipovetzky and Geffner 2012) to select which states to expand next. The novelty $w(s)$ of a state s is the size of the smallest non-empty set of ground atoms Q such that s is the first state visited by the search where $Q \subseteq s$. For example, if s is the first visited state containing atom P, then $w(s) = 1$. In contrast, if there is no single atom that first occurred in s but there is a subset $\{P_1, P_2\}$ that first occurred together in s, then $w(s) = 2$.

The simplest BFWS variant prioritizes in the open list those states with minimal w-value. However, the strongest BFWS planners apply novelty measures based on *partition functions* of the search space (Lipovetzky and Geffner 2017; Francès et al. 2017, 2018). The novelty $w_{\langle f_1,\ldots,f_n\rangle}(s)$ of a state s given functions $\langle f_1, \ldots, f_n\rangle$ is the size of the smallest set of atoms Q such that s is the first state visited where $Q \subseteq s$ among all states S where $f_i(s) = f_i(s')$ for $1 \leq i \leq n$ and for all $s' \in S$. In practice, these planners only evaluate novelty up to a bound k, where usually $k = 2$. If a state s has no novel tuple of size k or less, then $w(s) = k+1$.

An advantage of width-based search algorithms is that, *a priori*, the evaluation of w in a given state only depends on the state itself and the set of previously visited states. In other words, the evaluation of a state is *black-box* with respect to the structure of the problem. This makes width-based algorithms an attractive option for hard-to-ground tasks, where it is very expensive to obtain ground actions.

The main black-box BFWS algorithms use $w_{\langle \#r,\#g\rangle}(s)$ where the partition functions $\langle \#r, \#g\rangle$ define $\#r(s)$ as the number of *relevant atoms* that are true in s, and $\#g(s)$ as the number of goal atoms in γ that are true in s. Choosing the set of relevant atoms is a parameter of the search algorithm. There are several approaches for this choice point. In our work, we focus on the following two methods from the literature (Francès et al. 2017): *(i)* BFWS(R_0), where the set of relevant atoms is the empty set; and *(ii)* BFWS(R_X), where the set of relevant atoms is the set of *useful atoms* (Hoffmann and Nebel 2001) computed from a *relaxed plan* from s_0. An atom is considered useful for a state s if it appears in the effect of an action of a relaxed plan from s.

We study BFWS(R_0) because it is the baseline version of width-based search with simulators and it does not require any knowledge about the action structures. The choice of BFWS(R_X) is motivated by the work of Corrêa et al. (2020) who show how to extract a relaxed plan efficiently from the lifted representation. We do not consider the other methods from Francès et al. (2017) in the lifted setting because these either involve performing a "pre-search" (which is too expensive for hard-to-ground domains due to the lifted successor generation) or knowing all reachable atoms (which is prohibitive due to the expensive grounding).

Balancing Exploration and Exploitation

In some tasks, a pure novelty-guided search can be misleading, since it only focuses on exploring unseen parts of the state space, without exploiting any information about the structure of the problem. One way of making BFWS more goal-oriented is to combine it with *preferred operators*, i.e., operators that achieve useful atoms (Richter and Helmert 2009). To this end, we use BFWS inside a *dual-queue* approach, where the search keeps track of two open-list queues, as done by Richter and Helmert (2009). In the first queue q_1, we insert all generated states. In the second queue q_2, we insert only states reached via preferred operators. We use a *priority value* $p \in \mathbb{Z}$ and let the search expand a state from q_2 iff $p > 0$. The value of p is set to a constant C initially, we reduce it by 1 every time we select a state from q_2, and we *boost* it by C ($p := p + C$) whenever we expand a state s that contains more goal atoms than all states seen before s. We call this search algorithm *DQ-BFWS*.

By definition, useful atoms are state-dependent. Thus, we need to extract a relaxed plan for each state, which is equivalent to evaluating h^{add} (Bonet and Geffner 2001) in every state. To avoid this overhead, we compute a relaxed plan only for the initial state s_0 and consider the useful atoms of s_0 as useful for every state in the search. This is a relaxation on the definition of useful atoms but we hypothesize that this can still speed up the search.

Another approach for adding goal-direction to a BFWS search is to alternate between open-lists sorted by novelty measures and open-lists sorted by heuristics. Katz et al. (2017) show that alternating between open-lists (Röger and Helmert 2010) using only heuristics based on novelty evaluators can be beneficial. In our work, we run BFWS and alter-

nate between an open-list ordered by $w_{\langle \#r, \#g \rangle}$ and an open-list ordered by h^{add} or h^{FF} (Hoffmann and Nebel 2001).[1] Since we use h^{add} as one of the components, we can extract useful atoms for every state without further overhead. This goes in the opposite direction compared to our previous approach: we have the overhead of computing h^{add} for every state, but by alternating with a novelty-based queue, we hope to achieve a balance between exploration and exploitation and reduce the number of evaluated states. We denote this version as BFWS($[R, h^{\text{add}}]$), where R is the set of relevant atoms. For both evaluators R and h^{add}, BFWS($[R, h^{\text{add}}]$) also uses an open-list variant that only contains states reached via preferred operators. Thus, BFWS($[R, h^{\text{add}}]$) has four open-lists in total. It alternates between the open-lists of R and h^{add} each time it selects a new state to be expanded. For a given evaluator, the search then decides between the regular open-list or the open-list with only states reached via preferred operators based on the counter C, as explained above.

Implementation

Any implementation of a width-based search needs to keep track of the set of reached atoms. For $k = 1$, the straightforward implementation is to keep a bitmap where each position corresponds to a ground atom P, and the corresponding bit is set to 1 if P has been reached. To compute the novelty of a state s, we simply check if all atoms in the state have their corresponding bit set to 1. If not, then $w(s) = 1$ and we update the bitmap accordingly.

To generalize the computation for larger k, we can create a bitmap of size $\binom{n}{k}$, where n is the total number of ground atoms, and each entry corresponds to a tuple of ground atoms of size k. The entry for tuple Q is set to 1 iff Q has been achieved. The evaluation and the update of the bitmap is analogous to the $k = 1$ case. For the case of $w_{\langle \#r, \#g \rangle}$, the same idea still applies. The difference is that we need to create one bitmap for each $(\#r, \#g)$ pair.

Unfortunately, for hard-to-ground planning tasks, such an approach is usually infeasible. This is because it requires computing all reachable atoms in advance, which is often too expensive. Even when this computation is feasible, the number of atoms is often too large and thus creating bitmaps for all tuples of size k would add too much overhead.

To avoid grounding, we propose an alternative implementation based on the representation by Corrêa et al. (2020). In their representation, a state is a set of relations. For each relation, we associate each tuple with an index, similarly as for the bitmap. However, this indexing is done on-demand and an atom is indexed only once it is reached. We store a hash table that maps each reached atom to an index. To check if an atom P has been seen and indexed before, it suffices to check if there is an entry in the hash table with key P. If not, we add the entry for P mapping it to a fresh index value.

For $k = 1$, we evaluate $w(s)$ by checking if each atom in s has an entry in the hash table just described. For larger values of k, we keep track of the reached tuples Q by storing the indices of the atoms in Q in a set. In detail, to check if a tuple Q is reached for the first time, we first obtain the tuple Q' of all indices of atoms in Q. Then, we check if Q' is in the set and if not, we know that the state is novel and add Q' to the set. We use a different set for each value of k.

Both the bitmap used in the ground version and the data structures used in our implementation (i.e., the hash table used for indexation, and the set of tuples of indices) have an access time of $O(1)$. Since our data structures have far more overhead compared to the bitmap approach, it is important to use an efficient implementation. We also use a common optimization for width-based planners (Francès et al. 2017): if applying action A in state s yields state s', where $\#r(s) = \#r(s')$ and $\#g(s) = \#g(s')$, then we only consider tuples that contain an atom in $\mathit{add}(A)$ to evaluate $w_{\langle \#r, \#g \rangle}(s')$.

Experiments

We implemented all search algorithms in the Powerlifted planning system (Corrêa et al. 2020) and ran experiments with the Lab toolkit (Seipp et al. 2017) on Intel Xeon Silver 4114 processors running at 2.2 GHz. We use a time limit of 30 minutes and a memory limit of 16 GiB per task. Our primary benchmarks are the 862 hard-to-ground (HTG) tasks used by Lauer et al. (2021). This HTG set is divided into 8 domains. We also use a second benchmark set (IPC) that contains all 941 STRIPS tasks over 28 domains from previous IPCs, which are supported by both the Powerlifted and the FS-blind planner (Francès et al. 2018). We consider all action schemas as unit-cost. Our source code and experiment data are available online (Corrêa and Seipp 2022).

Comparison to the Ground Implementation

In the first experiment, we compare our lifted BFWS(R_0) implementation to the corresponding ground implementation. For the ground version, we use the FS-blind planner, which participated in the IPC 2018 (Francès et al. 2017, 2018). In both cases, we use $k = 2$. For the IPC set, the lifted implementation is on par with the ground version: the lifted version solves 714 tasks and has higher coverage in 10 domains, while the ground version solves 711 tasks and has higher coverage in 7 domains. We find these results remarkable, since our implementation is tailored for large tasks and we expected the bitmap representation to be superior for the smaller problems. For the HTG set, the lifted version is preferable, solving more tasks than the ground version in 5 of the 8 commonly supported domains (see Table 1).[2]

Comparison to Other Methods

We now compare the lifted implementations of the different algorithms described above to state-of-the-art ground and lifted planners, focusing on the HTG benchmark set.

As baselines for ground planners, we use *(i)* LAMA (Richter and Westphal 2010), which uses the FF heuristic and landmarks; and *(ii)* Dual-BFWS (Lipovetzky and Geffner 2017; Francès et al. 2018), a state-of-the-art width-based planner. For lifted planners, we compare to three approaches: *(i)* lazy greedy best-first search (GBFS)

[1] We introduce the algorithm in terms of h^{add} but it is analogous for h^{FF}.

[2] Since the FS planner does not support predicates with arity higher than 4, it cannot handle the visitall-5-dim instances.

	Baselines						Lifted BFWS				
	FS-blind	LAMA	Dual-BFWS	L-$h^{\text{gc, ur-d}}$	L-h^{add}	L-h^{FF}	R_0	R_X	DQ(R_X)	$[R_X, h^{\text{add}}]$	$[R_X, h^{\text{FF}}]$
IPC (1001)	714	917	**953**	575	762	821	725	741	736	838	857
blocksworld (40)	0	12	4	7	5	9	6	5	3	**21**	19
childsnack (144)	73	**116**	109	98	81	72	60	67	65	100	101
genome-edit-dist. (312)	**312**	**312**	**312**	**312**	285	311	307	**312**	**312**	309	309
logistics (40)	0	36	4	0	**40**	**40**	10	31	31	**40**	**40**
organic-synthesis (56)	0	21	20	47	47	48	48	49	49	**50**	**50**
pipesworld-tankage (50)	18	18	18	10	32	27	43	47	47	**48**	47
rovers (40)	2	16	13	16	31	**40**	0	1	1	**40**	**40**
visitall-multidim. (120)	37	60	36	100	100	98	108	111	**116**	101	101
visitall-5-dim (60)	–	12	6	**51**	42	42	48	48	**51**	42	41
HTG Total (862)	442	603	522	641	663	687	630	671	675	**751**	748

Table 1: Number of solved tasks by different planners on the IPC (summarized) and HTG benchmark sets.

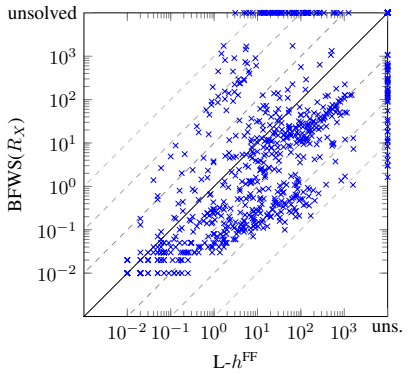

Figure 1: Runtime in seconds for BFWS(R_X) and L-h^{FF} on the HTG benchmark set.

(Doran and Michie 1966; Helmert 2006) using h^{add} and preferred operators by Corrêa et al. (2021), denoted as L-h^{add}; *(ii)* the same lazy GBFS but using h^{FF} and preferred operators (Corrêa et al. 2022), denoted as L-h^{FF}; and *(iii)* the goal–count heuristic (Fikes and Nilsson 1971) using the unary relaxation heuristic as tiebreaker, which is the best method from Lauer et al. (2021), denoted as L-$h^{\text{gc, ur-d}}$.

Table 1 shows overall coverage for the IPC set and per-domain coverage for the HTG set. We only analyze results for the HTG set in detail below. We first compare BFWS(R_0) and BFWS(R_X) to the baselines. Both planners use $k = 2$, which always yields higher coverage than $k = 1$ (not shown). Both approaches are superior to the ground methods in several HTG domains and competitive with the other lifted baselines. In fact, BFWS(R_X) outperforms L-h^{add} and L-$h^{\text{gc, ur-d}}$ in total coverage on the HTG set.

Since L-h^{FF} is guided by a stronger heuristic than BFWS(R_X) and since L-h^{FF} uses a lazy search with preferred operators, L-h^{FF} frequently needs fewer state evaluations than BFWS(R_X). Yet, as Figure 1 shows, BFWS(R_X) usually needs less time to solve tasks, which is due to BFWS(R_X) evaluating states faster than L-h^{FF}.

Finally, we analyze how our new algorithms, DQ-BFWS(R_X), BFWS($[R_X, h^{\text{add}}]$), and BFWS($[R_X, h^{\text{FF}}]$) perform on the HTG set. For DQ-BFWS(R_X), we report results for $k = 2$, while for BFWS($[R_X, h^{\text{add}}]$) and BFWS($[R_X, h^{\text{FF}}]$) we show results for $k = 1$ as it yields higher overall coverage. We use $C = 1000$ for all configurations. Inspecting Table 1, we see that all three planners yield a strong performance. DQ-BFWS(R_X) solves roughly as many tasks as BFWS(R_X), its single-queue counterpart, and it is competitive with L-h^{FF}, the state-of-the-art lifted planner in the literature. BFWS($[R_X, h^{\text{add}}]$) and BFWS($[R_X, h^{\text{FF}}]$) solve roughly the same number of tasks across all domains. The two planners obtain a much higher total coverage than all other planners (751 and 748 tasks), showing that it is indeed beneficial to make DQ-BFWS(R_X) more goal directed by combining it with L-h^{add} or L-h^{FF}. In six domains, BFWS($[R_X, h^{\text{add}}]$) solves at least as many tasks as the stronger of the two ingredient planners for that domain. We also see that DQ-BFWS(R_X) is quite complementary to BFWS($[R_X, h^{\text{add}}]$): they obtain the highest coverage among all evaluated planners in three and five domains, respectively, and only in the childsnack domain a different planner is preferable to both of them.

Conclusions & Future Work

In this work, we investigated how BFWS performs in hard-to-ground domains. We showed that the traditional ground representation used by existing planners is not competitive with other lifted methods, but that a re-implementation of BFWS, taking into account the lifted representation, reaches state-of-the-art performance. We also presented ways to make the search more informed by using different evaluators and preferred operators together with the novelty criteria. In this manner, we enhanced the exploratory behavior of BFWS with the exploitative behavior of the h^{add} and h^{FF} heuristics. In our experiments, we showed that the novelty measures have high synergy with both heuristics, increasing the number of solved tasks significantly.

There are several other BFWS-based algorithms that can be implemented using our new representation (e.g., Katz et al. 2017). For more sophisticated BFWS variants, such as BFWS(f_6) (Lipovetzky and Geffner 2017), we will need to adapt landmarks to the lifted setting (Wichlacz, Höller, and Hoffmann 2021).

Acknowledgments

We have received funding for this work by the Swiss National Science Foundation (SNSF) as part of the project "Certified Correctness and Guaranteed Performance for Domain-Independent Planning" (CCGP-Plan). Moreover, this work was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215, and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. We thank Guillem Francès for clarifying some implementation details of the FS-blind planner.

References

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence*, 129(1): 5–33.

Corrêa, A. B.; Francès, G.; Pommerening, F.; and Helmert, M. 2021. Delete-Relaxation Heuristics for Lifted Classical Planning. In *Proc. ICAPS 2021*, 94–102.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation using Query Optimization Techniques. In *Proc. ICAPS 2020*, 80–89.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2022. The FF Heuristic for Lifted Classical Planning. In *Proc. AAAI 2022*.

Corrêa, A. B.; and Seipp, J. 2022. Code and experiment data from the ICAPS 2022 paper "Best-First Width Search for Lifted Classical Planning". https://doi.org/10.5281/zenodo.6373934.

Doran, J. E.; and Michie, D. 1966. Experiments with the Graph Traverser program. *Proceedings of the Royal Society A*, 294: 235–259.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2: 189–208.

Francès, G.; Geffner, H.; Lipovetzky, N.; and Ramiréz, M. 2018. Best-First Width Search in the IPC 2018: Complete, Simulated, and Polynomial Variants. In *IPC-9 Planner Abstracts*, 23–27.

Francès, G.; Ramírez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely Declarative Action Representations are Overrated: Classical Planning with Simulators. In *Proc. IJCAI 2017*, 4294–4301.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*, volume 13 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool.

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

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial Intelligence*, 173: 503–535.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Katz, M.; Lipovetzky, N.; Moshkovich, D.; and Tuisov, A. 2017. Adapting Novelty to Classical Planning as Heuristic Search. In *Proc. ICAPS 2017*, 172–180.

Lauer, P.; Torralba, Á.; Fišer, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In *Proc. IJCAI 2021*.

Lipovetzky, N.; and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In *Proc. ECAI 2012*, 540–545.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *Proc. AAAI 2017*, 3590–3596.

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

Richter, S.; and Helmert, M. 2009. Preferred Operators and Deferred Evaluation in Satisficing Planning. In *Proc. ICAPS 2009*, 273–280.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Ridder, B.; and Fox, M. 2014. Heuristic Evaluation Based on Lifted Relaxed Planning Graphs. In *Proc. ICAPS 2014*, 244–252.

Röger, G.; and Helmert, M. 2010. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In *Proc. ICAPS 2010*, 246–249.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Wichlacz, J.; Höller, D.; and Hoffmann, J. 2021. Landmark Heuristics for Lifted Planning – Extended Abstract. In *Proc. SoCS 2021*, 242–244.