Consolidating LAMA with Best-First Width Search

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

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

Abstract

One key decision for heuristic search algorithms is how to balance exploration and exploitation. In classical planning, novelty search has come out as the most successful approach in this respect. The idea is to favor states that contain previously unseen facts when searching for a plan. This is done by maintaining a record of the tuples of facts observed in previous states. Then the novelty of a state is the size of the smallest previously unseen tuple. The most successful version of novelty search is best-first width search (BFWS), which combines novelty measures with heuristic estimates. An orthogonal approach to balance exploration-exploitation is to use several open-lists. These open-lists are ordered using different heuristic estimates, which diversify the information used in the search. The search algorithm then alternates between these open-lists, trying to exploit these different estimates. This is the approach used by LAMA, a classical planner that, a decade after its release, is still considered state-of-the-art in agile planning. In this paper, we study how to combine LAMA and BFWS. We show that simply adding the strongest open-list used in BFWS to LAMA harms performance. However, we show that combining only parts of each planner leads to a new state-of-the-art agile planner.

Introduction

Agile planning involves solving planning tasks as fast as possible, with little or no consideration for plan quality. While this might not be suitable for some domains, it is still an important question in general, and it is closely related to the problem of deciding plan existence (e.g., Bylander 1994). Since 2014, there have been dedicated tracks for agile planning in the classical part of the International Planning Competition (IPC). Usually, planners are given a 5 minutes per task, and they are scored based on how quickly they solve the task.

In the recent IPC 2023, LAMA (Richter, Westphal, and Helmert 2011), a 15 year-old planner that was included as a baseline, obtained a higher total agile score than all competitors, including the winner DecStar (Gnad, Torralba, and Shleyfman 2023).[1] A similar situation occurred in the IPC 2018, where the LAMA baseline scored second place.

LAMA uses an *alternation of open-lists* (Röger and Helmert 2010) together with *deferred evaluation* and *preferred operators* (Richter and Helmert 2009) to find a plan quickly. This alternation allows LAMA to balance between multiple heuristic estimates and allowed it to win the Satisficing Track of the IPC in 2008. Even though the LAMA code base is continuously improved, the core behavior remains unchanged. This raises the question of how we can advance the state of the art and obtain a planner that finds plans faster than LAMA.

In the IPC 2018 Agile Track, the only planner to outperform LAMA was based on best-first width search (BFWS) (Lipovetzky and Geffner 2017; Francès et al. 2018). The idea of BFWS is to expand states that are *novel* (Lipovetzky and Geffner 2012). There are multiple ways to define the novelty of a state (e.g., Katz et al. 2017), but the key idea is that the state contains a tuple of atoms which has not been encountered before by the search. When combined with heuristic estimates, BFWS achieves a good exploration-exploitation trade-off which helps find plans quickly.

In this paper, we aim to combine the advantages of LAMA and BFWS. Our basic idea is very simple: we add the open-list used by BFWS (more precisely, BFWS(f_6)) to LAMA, as one of its alternated open-lists. One could expect that this combination would result in a faster planner because classical planning has a history of "*the more, the merrier*": more open-lists (e.g., Röger and Helmert 2010; Corrêa et al. 2023), more heuristics (e.g., Franco et al. 2017; Seipp, Keller, and Helmert 2020), more tie-breakers (e.g., Asai and Fukunaga 2017b), more planners (e.g., Helmert, Röger, and Karpas 2011; Cenamor, de la Rosa, and Fernández 2016), all seem to improve coverage of the planners.

It turns out though, that this combination actually yields a *worse* planner than LAMA and BFWS(f_6) alone. However, a detailed ablation study reveals that *removing features* from LAMA and BFWS drastically speeds up the planner. In fact, by removing entire open-lists of the LAMA configuration, and discarding function estimates and tie-breakers from BFWS(f_6), improves the coverage and IPC agile score compared to the original systems. The resulting planner also outperforms all agile competitors of the last IPCs by a large margin, with an IPC agile score 13% higher than the second best method. Overall, our new method establishes a new state of the art for agile planning.

[1]When we mention LAMA in our paper, we refer to the *first iteration* of LAMA, which is responsible for all its coverage. Further iterations are used to improve plan quality.

Background

A *state space* is a tuple $\mathcal{S} = \langle S, s_I, G, succ\rangle$, where S is a set of *states*, $s_I \in S$ is the *initial state*, G is the *goal description*, *succ* is a *successor function* mapping each state to a finite (possible empty) set of successor states. A state is a set of *facts*. If a state s contains fact f or a set of facts F, we say that s *satisfies* f or F. It is sufficient to consider S as the minimal set where $s_I \in S$, and $succ(s) \subseteq S$ for each $s \in S$. The goal G is also a set of facts. A state s_* is a *goal state* if $G \subseteq s_*$.

A sequence of states $\tau = \langle s_0, \ldots, s_n\rangle$ is a *path* from s_0 to s_n in S if $s_i \in succ(s_{i-1})$ for $i \in \{1, \ldots, n\}$. Path τ is an *s-plan* if $s_0 = s$ and $G \subseteq s_n$ and a *plan* for $\mathcal{S}$ if $s_0 = s_I$.

We consider *agile planning*, the problem of computing plans as fast as possible, without caring about their *quality* (i.e., number of states visited by the plan).

A common method to find plans is via *heuristic search*. A *heuristic* is a function $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$. It estimates the length of an s-plan for states $s \in S$. Heuristic search algorithms start from s_I and explore states guided by some heuristic h, preferring states s with low $h(s)$ values. Examples of strong heuristics for agile planning are h^{add} (Bonet and Geffner 2001), h^{FF} (Hoffmann and Nebel 2001), and h^{LM}(Richter, Helmert, and Westphal 2008). We assume familiarity with common search algorithms such as greedy best-first search (Doran and Michie 1966).

Instead of being guided by a heuristic, BFS(w) selects states for expansion based on their *novelty*, preferring states with low w values (Lipovetzky and Geffner 2012, 2017). The novelty $w(s)$ of a state s is the size of the smallest set of facts F such that s is the first state visited that satisfies F.

This simple scheme can be turned into state-of-the-art *best-first width search* (BFWS) algorithms by extending it with *partition functions* (Lipovetzky and Geffner 2017; Francès et al. 2017, 2018). For BFWS, the novelty $w_{\langle h_1,\ldots,h_n\rangle}(s)$ of a state s given the *partition functions* $\langle h_1, \ldots, h_n\rangle$ is the size of the smallest set of facts F such that s is the first evaluated state that subsumes F, among all states s' visited before s for which $h_i(s) = h_i(s')$ for $1 \leq i \leq n$. 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 $w(s) = k + 1$.

Balancing Exploration and Exploitation

One important design choice of a planner is how it balances *exploration* and *exploitation*. Exploration techniques search parts of the state space that have not yet been visited. Exploitation techniques prefer going into parts that are considered more promising by some metric. For example, choosing the next expanded state at random is a form of exploration; choosing it based on a heuristic is exploitation.

Modern planners usually mix both of them. A common technique is to keep several open-lists during search, each one guided by a different heuristic (Röger and Helmert 2010). Some lists are even *incomplete* since they only retain a subset of the generated states, e.g., states generated via *preferred operators* (Hoffmann and Nebel 2001; Richter and Helmert 2009). The simplest yet most successful method for combining multiple open-lists is *alternation* (Helmert 2004, 2006). An alternation-based search algorithm maintains n open-lists, where the i-th open-list is ordered by some heuristic h_i. We denote it as $[h_1, \ldots, h_n]$. The search alternates between the open-lists in a round-robin fashion: first it expands the best state according to h_1, adds all successors to all (or some of the) open-lists, then it expands the best state according to h_2, and so on. In iteration $n + 1$ it expands from h_1 again.

Alternation is one of the main building blocks used by the LAMA planner (Richter and Westphal 2008, 2010; Richter, Westphal, and Helmert 2011). LAMA keeps four open-lists: $[h^{\text{FF}}, h_+^{\text{FF}}, h^{\text{LM}}, h_+^{\text{LM}}]$, where h_+ denotes an open-list ordered by h but only containing states reached via preferred operators. LAMA is based on the Fast Downward planning system (Helmert 2006) and won the IPC editions of 2008 and 2011. Until today, it is considered state-of-the-art in agile planning.

An orthogonal way of combining exploration and exploitation is by using *tiebreakers* (Röger and Helmert 2010). A tiebreaking open-list $\langle h_1, \ldots, h_n\rangle$ uses a ranking over n heuristics. It selects states based on h_i and, if there is a tie, breaks this tie using h_{i+1}. It keeps only a single open-list, but the order of this list is defined by multiple heuristics. Throughout the paper, we assume that if all $h_1, \ldots, h_n$ are tied, then remaining ties are broken by g-value. If ties persist, then we assume a FIFO ordering.

While LAMA opted for an alternation open-list, the more sophisticated versions of BFWS use tiebreaking (Lipovetzky and Geffner 2017). In general, BFWS(f) orders its open-list by $f = \langle f_1, \ldots, f_n\rangle$. The strongest version of BFWS is BFWS(f_6), where the open-list is ordered by $f_6 = \langle w_{\langle h^{\text{LM}}, h^{\text{FF}}\rangle}, pref, h^{\text{LM}}, w_{\langle h^{\text{FF}}\rangle}, h^{\text{FF}}\rangle$ where *pref* is an indicator function yielding 1 for states reached via a preferred operator. Simpler versions include BFWS(f_4), where $f_4 = \langle w_{\langle h^{\text{LM}}, h^{\text{FF}}\rangle}, h^{\text{LM}}, h^{\text{FF}}\rangle$, and BFWS($f_2$), where $f_2 = \langle w_{\langle h^{\text{FF}}\rangle}, h^{\text{FF}}\rangle$.

A BFWS-based planner, BFWS-Preference, won the Agile Track of the IPC 2018 (Francès et al. 2018). This planner is the only one to achieve a higher agile score than LAMA in any of the IPCs, which is remarkable given that LAMA only served as a baseline planner.

Combining LAMA and BFWS

LAMA and BFWS present distinct ways of combining exploration and exploitation. But they do have more similarities than initially meets the eye. If we consider BFWS(f_6), then both planners use exactly the same information (with the exception of the novelty measures): h^{FF}, h^{LM}, and preferred operators. The difference is in the way the planners process the information and the question is how we can combine the advantages of both in a single planner. Arguably the most direct approach to combine both planners is to use the tiebreaking open-list of BFWS(f_6) as an additional open list in LAMA. We call this modification LAMA-W(f_6).

Before we evaluate this new planner, we present the experimental setup used throughout the paper. We implemented LAMA-W(f_6) and BFWS(f_6), alongside the sim-

	LAMA	BFWS(f_6)	LAMA-W(f_6)	LAMA-W(f_4)	LAMA-W($f_2^{h^{\mathrm{FF}}}$)	LAMA-W($f_2^{h^{\mathrm{LM}}}$)	LAMA-W($w_{\langle h^{\mathrm{FF}}\rangle}$)	LAMA-W($w_{\langle h^{\mathrm{LM}}\rangle}$)	LAMA-W($w_{\langle\rangle}$)
Coverage	2081	2042	2029	2037	2047	**2113**	1957	2096	2094
Expansions Score	1629.31	**1687.50**	1359.53	1363.68	1345.63	1643.34	1238.47	1630.12	1634.12
Agile Score	1737.43	1581.45	1593.69	1600.41	1587.31	**1751.11**	1509.52	1740.41	1739.66

Table 1: Scores for the baselines, LAMA, BFWS(f_6) and LAMA-W(f_6), and for simplifications of LAMA-W(f_6).

pler BFWS variants, within the Scorpion planning system (Seipp, Keller, and Helmert 2020), which is an extension of Fast Downward (Helmert 2006). All our alternation-based methods use the default boosting parameters of LAMA (Richter and Westphal 2010): whenever the search encounters a state s with a new lowest heuristic value $h(s)$ for any heuristic h, all preferred operator queues receive a *boost* value of 1000. This incurs that for the next $1000 \cdot n$ expansions only states from preferred operator queues are considered, where n is the number of such queues.

For running our experiments, we use Downward Lab (Seipp et al. 2017) on AMD EPYC 7742 processors running at 2.25 GHz. We use the same limits as the IPC 2023 Agile Track: a 5 minutes time limit, and 8 GiB of memory per task. Our benchmark set consists of 2426 tasks from all IPCs (1998–2023). We omit domains containing derived predicates, as some of the planners do not support them. We also use the h^2-preprocessor for all planner runs (Alcázar and Torralba 2015). We employ three main scores to evaluate planner performance: coverage score,[2] expansions score, and agile score.[3] The total score is the sum over all tasks.

The first three columns of Table 1 compare LAMA and BFWS(f_6) with LAMA-W(f_6). We see that the new algorithm solves fewer tasks and needs more expansions than both LAMA and BFWS(f_6) in almost all metrics, but its agile score is slightly higher than the one by BFWS(f_6). The low expansions score indicates that the addition of the new open-list makes the search less informed. This is somewhat counter-intuitive because in many cases combining different evaluators within the same search algorithm decreases the number of expansions (c.f., Corrêa et al. 2023).

To understand what exactly has such a negative impact on performance, we decomposed the features of LAMA-W(f_6). Our first step is to iteratively simplify the new open list by removing its subsets of its features (partition functions, tie-breakers). We define the following variations:

1. LAMA-W(f_4), where $f_4 = \langle w_{\langle h^{\mathrm{LM}}, h^{\mathrm{FF}}\rangle}, h^{\mathrm{LM}}, h^{\mathrm{FF}}\rangle$;
2. LAMA-W(f_2^h). where $f_2^h = \langle w_{\langle h\rangle}, h\rangle$. We tested two different versions: $h = h^{\mathrm{FF}}$ and $h = h^{\mathrm{LM}}$;
3. LAMA-W($w_{\langle h\rangle}$) uses the open-list of BFWS($w_{\langle h\rangle}$) (Lipovetzky and Geffner 2017), a best-first width search with a single open-list ordered by $w_{\langle h\rangle}$ and breaking ties by accumulated cost g. When $h = \langle\rangle$ we do not use any partition function.

[2]The coverage score is 1 for solved tasks and 0 otherwise.

[3]The expansions score is based on the number of expanded states, while the agile score is based on the runtime (Richter and Helmert 2009). Performance better than a lower bound (100 states for expansions and 1 second for runtime) counts as 1. Performance worse than an upper bound U (10^6 states for expansions and 300s for runtime) counts as 0. We interpolate intermediate values with a logarithmic function: $1 - {\log(x)}/{\log(U)}$ where x is the number of expansions/runtime in seconds.

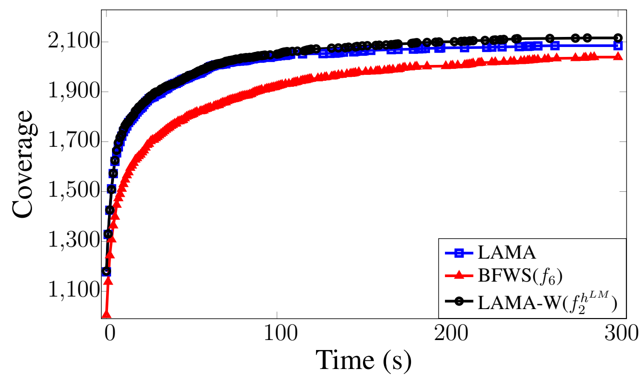

Figure 1: Coverage over time for LAMA, BFWS(f_6), and LAMA-W($f_2^{h^{\mathrm{LM}}}$).

Table 1 shows the results for all methods. From left to right, the novelty-based open list becomes increasingly simple. There are two immediate observations to be made: first, simpler approaches, such as LAMA-W($f_2^{h^{\mathrm{LM}}}$) and LAMA-W($w_{\langle\rangle}$), have much better coverage and agile score than LAMA-W(f_6); second, using h^{LM} in the novelty open-list is consistently superior to h^{FF}. LAMA-W($f_2^{h^{\mathrm{LM}}}$) improves over LAMA and LAMA-W(f_6) in all measured criteria. It is also superior to BFWS(f_6) in coverage and agile scores.

Figure 1 shows the coverage over time for LAMA, BFWS(f_6), and LAMA-W($f_2^{h^{\mathrm{LM}}}$). Albeit having a better expansions score, BFWS(f_6) is slower than the other two methods. LAMA and LAMA-W($f_2^{h^{\mathrm{LM}}}$) run similarly fast, and the difference in the curves is only perceptible when very close to the time limit. This indicates that LAMA-W($f_2^{h^{\mathrm{LM}}}$) has an edge over LAMA in larger tasks, where the extra exploration added by the novelty open-list helps.

Consolidating the LAMA Open-Lists

We started from LAMA-W(f_6) and boiled the novelty-based open list down to obtain LAMA-W($f_2^{h^{\mathrm{LM}}}$), which then outperformed both LAMA and BFWS(f_6). But who says that

															NOLAN		LAMA-W($f_2^{h^{\text{LM}}}$)
h^{FF}									✓	✓	✓	✓	✓	✓	✓	✓
h_{+}^{FF}					✓	✓	✓	✓					✓	✓	✓	✓
h^{LM}			✓	✓			✓	✓			✓	✓			✓	✓
h_{+}^{LM}		✓		✓		✓		✓		✓		✓		✓		✓
Coverage	1668	2001	1803	2002	2081	2112	2081	**2123**	2023	2056	2013	2052	2076	2120	2074	2113
Exp. Score	1029.7	1449.3	1113.0	1444.2	1533.4	1631.6	1536.7	1642.5	1332.9	1502.6	1333.8	1499.5	1539.6	**1644.0**	1535.8	1643.3
Agile Score	1278.6	1632.3	1380.7	1627.8	1694.8	1744.3	1699.9	1753.9	1581.1	1687.2	1577.7	1681.2	1699.5	**1756.3**	1696.5	1751.1

Table 2: Results for the consolidation of LAMA open-lists in LAMA-W($f_2^{h^{\text{LM}}}$) (last column). The first four rows correspond to the four original open-lists in LAMA and the active open-lists are marked with ✓. All configurations include the $f_2^{h^{\text{LM}}}$ open-list.

	LAMA	NOLAN	Diff.
barman-sat14 (20)	19	20	+1
caldera-split-sat18 (20)	6	10	+4
cavediving-14 (20)	7	8	+1
data-network-sat18 (20)	11	15	+4
depot (22)	20	22	+2
freecell (80)	79	80	+1
logistics98 (35)	34	35	+1
maintenance-sat14 (20)	11	12	+1
nomystery-sat11 (20)	13	17	+4
nurikabe-sat18 (20)	16	17	+1
pathways (30)	23	25	+2
pipesworld-notankage (50)	44	45	+1
pipesworld-tankage (50)	41	42	+1
ricochet-robots-sat23 (20)	6	14	+8
rubiks-cube-sat23 (20)	20	17	-3
snake-sat18 (20)	4	9	+5
storage (30)	19	26	+7
tetris-sat14 (20)	17	20	+3
thoughtful-sat14 (20)	16	18	+2
tidybot-sat11 (20)	18	20	+2
transport-sat11 (20)	15	18	+3
transport-sat14 (20)	11	12	+1
trucks (30)	15	17	+2
trucks-strips (30)	14	16	+2
visitall-sat11 (20)	20	19	-1
visitall-sat14 (20)	20	4	-16
Other 55 domains (1729)	1562	1562	0
Total (2426)	2081	**2120**	**+39**

Table 3: Per domain coverage for LAMA and NOLAN. We only show domains where coverage differs. The last column shows the coverage gain/loss for NOLAN.

all original parts of LAMA are beneficial in a planner with a novelty-based open list? To evaluate this, we now consolidate the remaining parts of LAMA-W($f_2^{h^{\text{LM}}}$) by removing subsets of LAMA open-lists from LAMA-W($f_2^{h^{\text{LM}}}$).

As explained above, the original LAMA planner uses four open-lists: $[h^{\text{FF}}, h_{+}^{\text{FF}}, h^{\text{LM}}, h_{+}^{\text{LM}}]$, while LAMA-W($f_2^{h^{\text{LM}}}$) has

	Coverage	Agile Score
NOLAN	**2120**	**1756.3**
GBFS-RSL	2049	1557.3
Dual-BFWS	1986	1537.4
OLCFF	2024	1504.9
BFWS-Pref.	1927	1500.7
MERWIN	1926	1463.0
Stone Soup	2050	1450.6
DecStar	1962	1437.3

Table 4: Scores for state-of-the-art planners.

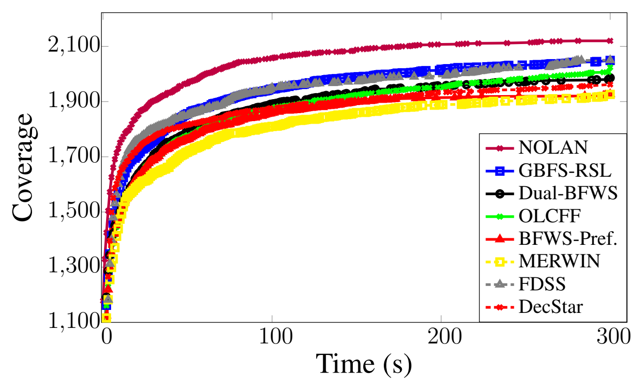

Figure 2: Coverage over time for each state-of-the-art planner and NOLAN.

five: $[h^{\text{FF}}, h_{+}^{\text{FF}}, h^{\text{LM}}, h_{+}^{\text{LM}}, f_2^{h^{\text{LM}}}]$. Table 1 shows the effect of removing combinations of of the original 4 queues in LAMA from LAMA-W($f_2^{h^{\text{LM}}}$). The two strongest versions use three open-lists from LAMA plus the $f_2^{h^{\text{LM}}}$ open-list. In comparison to LAMA-W($f_2^{h^{\text{LM}}}$), removing the h^{FF} open-list yields the highest coverage, while removing the h^{LM} open-list improves expansions and agile scores. Interestingly, the open-lists containing only preferred operators (h_{+}^{FF} and h_{+}^{LM}) seem to be the most essential ones in the planner.

For the rest of the paper, we use the version from Table 2 that has the highest agile and expansions scores (third column from the right). It alternates between

$[h^{\text{FF}}, h_{+}^{\text{FF}}, h_{+}^{\text{LM}}, f_2^{h^{\text{LM}}}]$. We call it NOLAN because it uses (among other features) NOvelty, Landmarks and AlternatioN. The accumulated coverage over time by NOLAN is almost identical LAMA-W($f_2^{h^{\text{LM}}}$) in Figure 1.

To better compare NOLAN to LAMA, we now analyze their results per domain. Table 3 shows the coverage for the 26 domains where the two planners solve a different number of tasks. In 23 out of the 26, NOLAN solves more tasks than LAMA. In more than half of these 23 domains, NOLAN solves two tasks or more compared to LAMA. On the flip side, LAMA has a higher coverage in only 3 domains. Two of them contain VisitAll tasks, where an agent has to visit all cells in a rectangular grid. For these tasks, NOLAN frequently runs out of memory. This is because VisitAll tasks have a huge number facts and different h^{LM} values, and the planner maintains a bit vector storing the seen fact tuples for each h^{LM} value. BFWS-based planners address this problem with a simple strategy: if the estimated memory usage for storing the seen fact tuples is larger than 2 GiB, they fall back to using $w = 1$. We leave addressing the memory bottleneck for such tasks as future work.

Related Work

Before we compare NOLAN to state-of-the-art agile planners, we discuss related work, some of which forms the basis for some of the planners we compare to.

Balancing exploration and exploitation is a longstanding challenge in classical planning (Hoffmann and Nebel 2001; Richter and Westphal 2008; Nakhost and Müller 2009; Vidal 2011; Katz et al. 2017; Asai and Fukunaga 2017a; Fickert 2018). In recent years, BFWS has emerged as the most successful approach for this problem (Lipovetzky and Geffner 2012, 2017; Francès et al. 2017). We evaluate two representatives, BFWS-Preference and Dual-BFWS (Francès et al. 2018), in the next section.

Follow-up work combined BFWS with other search techniques. For example, Katz et al. (2017) combine the concept of novelty with heuristic estimates, extending the definition of novelty by Shleyfman, Tuisov, and Domshlak (2016) to take into account the heuristic value of the states. This allows them to quantify how novel a state is, so the search can be guided directly by this value. However, as this metric is not goal-aware, Katz et al. need to break ties using traditional goal-aware functions. The MERWIN planner (Katz et al. 2018), that we evaluate below, is based on these ideas.

Another successful approach is due to Fickert (2020), who uses an orthogonal approach to the one by Katz et al.: instead of using novelty as the main guidance for the search, Fickert uses traditional heuristics to guide a greedy best-first search, and uses a lookahead strategy to find states with lower heuristic values quickly. This lookahead strategy is designed to reach states satisfying relaxed subgoals (Lipovetzky and Geffner 2014). To make the procedure efficient, he uses novelty pruning (Lipovetzky and Geffner 2012; Fickert 2018) to reduce the number of evaluated states. Fickert shows that there is a synergy between the novelty-based lookahead and the $h^{C\text{FF}}$ heuristic (Fickert and Hoffmann 2017), as the result from the lookahead can be used to trigger the refinement procedure of $h^{C\text{FF}}$. This idea was also used in the OLCFF planner (Fickert and Hoffmann 2018) from the IPC 2018. We include OLCFF and an improved version of it, GBFS-RSL, in our empirical comparison below.

A simpler (but still efficient) idea was introduced by Corrêa and Seipp (2022) in the context of lifted planning. Besides showing how to compute novelty over a lifted representation (where reachable facts are not known in advance), they also introduce a planner that alternates between a novelty-based open list and an open list ordered by traditional heuristic estimates. While their planner has an overhead in most IPC domains due to the first-order representation used, it is competitive with other planners for larger tasks.

While many successful idea in classical planning follow the "*the more, the merrier*" idea (Röger and Helmert 2010), some lines of research go in the opposite direction. For example, Tuisov and Katz (2021) define *novel* and *non-novel operators*, and show that pruning non-novel operators in preferred queues can increase coverage.

State-of-the-Art Agile Planners

We now compare NOLAN to state-of-the-art agile planners which scored highly in previous IPCs: BFWS-Preference (Francès et al. 2018), the winner of the Agile Track of IPC 2018; Dual-BFWS (Francès et al. 2018), another BFWS-based participant of IPC 2018; MERWIN (Katz et al. 2018), a planner that combines the state-of-the-art satisficing planner Mercury (Katz and Hoffmann 2014) with novelty heuristics (Katz et al. 2017); OLCFF (Fickert and Hoffmann 2018), which combines novelty pruning with the h^{CFF} heuristic (Hoffmann and Fickert 2015); DecStar (Gnad, Torralba, and Shleyfman 2023), the winner of the Agile Track at the IPC 2023; and Fast Downward Stone Soup 2023 (Büchner et al. 2023), the runner-up in the IPC 2023 Agile Track. Furthermore, we also include GBFS-RSL (Fickert 2020), which is an improved version of the OLCFF planner.

Table 4 compares NOLAN with all state-of-the-art planners.[4] NOLAN has the highest coverage and agile score. The relative difference in agile score to the second best planner, GBFS-RSL, is about 13%, while the difference to the previous winners of the IPC Agile Tracks is about 15%. As mentioned above, no other planner has an agile score close to LAMA, even though some planners, e.g., Fast Downward Stone Soup 2023, use LAMA as one of their components.

Figure 2 shows the coverage over time for all of these planners. The gap between NOLAN and all other methods is clearly visible. After 100 seconds, NOLAN already solves more tasks than any other method at any point in time.

Conclusions

We showed how to combine two successful agile planners, LAMA and BFWS(f_6). The combination of both did not meet with success, and it yielded a worse planner than any of its ingredients. But by carefully removing features from

[4]We do not compare expansion scores for this experiment because some planners perform multiple searches and some use lookaheads, which skews the total number of expansions.

LAMA and the BFWS open-list, we obtained state-of-the-art performance. The resulting planner NOLAN has a higher coverage and agile score than all other evaluated planners. These results raise the question of what happens if we combine NOLAN with other exploration techniques (e.g., Xie et al. 2014).

Acknowledgments

Augusto B. Corrêa was funded by the Swiss National Science Foundation (SNSF) as part of the project "Lifted and Generalized Representations for Classical Planning" (LGR-Plan). Jendrik Seipp was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Moreover, this work was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under grant agreement no. 952215.

References

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Asai, M.; and Fukunaga, A. 2017a. Exploration Among and Within Plateaus in Greedy Best-First Search. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 11–19. AAAI Press.

Asai, M.; and Fukunaga, A. 2017b. Tie-Breaking Strategies for Cost-Optimal Best First Search. *Journal of Artificial Intelligence Research*, 58: 67–121.

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

Büchner, C.; Christen, R.; Corrêa, A. B.; Eriksson, S.; Ferber, P.; Seipp, J.; and Sievers, S. 2023. Fast Downward Stone Soup 2023. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69(1–2): 165–204.

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2016. The IBaCoP Planning System: Instance-Based Configured Portfolios. *Journal of Artificial Intelligence Research*, 56: 657–691.

Corrêa, A. B.; Francès, G.; Hecher, M.; Longo, D. M.; and Seipp, J. 2023. Scorpion Maidu: Width Search in the Scorpion Planning System. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Corrêa, A. B.; and Seipp, J. 2022. Best-First Width Search for Lifted Classical Planning. In Thiébaux, S.; and Yeoh, W., eds., *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 11–15. AAAI Press.

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

Fickert, M. 2018. Making Hill-Climbing Great Again through Online Relaxation Refinement and Novelty Pruning. In Bulitko, V.; and Storandt, S., eds., *Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS 2018)*, 158–162. AAAI Press.

Fickert, M. 2020. A Novel Lookahead Strategy for Delete Relaxation Heuristics in Greedy Best-First Search. In Beck, J. C.; Karpas, E.; and Sohrabi, S., eds., *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, 119–123. AAAI Press.

Fickert, M.; and Hoffmann, J. 2017. Complete Local Search: Boosting Hill-Climbing through Online Relaxation Refinement. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 107–115. AAAI Press.

Fickert, M.; and Hoffmann, J. 2018. OLCFF: Online-Learning h^{CFF}. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 17–19.

Francès, G.; Geffner, H.; Lipovetzky, N.; and Ramiréz, M. 2018. Best-First Width Search in the IPC 2018: Complete, Simulated, and Polynomial Variants. In *Ninth International Planning Competition (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 Sierra, C., ed., *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, 4294–4301. IJCAI.

Franco, S.; Torralba, Á.; Lelis, L. H. S.; and Barley, M. 2017. On Creating Complementary Pattern Databases. In Sierra, C., ed., *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, 4302–4309. IJCAI.

Gnad, D.; Torralba, Á.; and Shleyfman, A. 2023. DecStar-2023. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Helmert, M. 2004. A Planning Heuristic Based on Causal Graph Analysis. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 161–170. AAAI Press.

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

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A Baseline for Building Planner Portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, 28–35.

Hoffmann, J.; and Fickert, M. 2015. Explicit Conjunctions w/o Compilation: Computing $h^{\mathrm{FF}}(\Pi^C)$ in Polynomial Time. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 115–119. AAAI Press.

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.; and Hoffmann, J. 2014. Mercury Planner: Pushing the Limits of Partial Delete Relaxation. In *Eighth International Planning Competition (IPC-8): Planner Abstracts*, 43–47.

Katz, M.; Lipovetzky, N.; Moshkovich, D.; and Tuisov, A. 2017. Adapting Novelty to Classical Planning as Heuristic Search. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 172–180. AAAI Press.

Katz, M.; Lipovetzky, N.; Moshkovich, D.; and Tuisov, A. 2018. MERWIN Planner: Mercury Enchanced With Novelty Heuristic. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 53–56.

Lipovetzky, N.; and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 540–545. IOS Press.

Lipovetzky, N.; and Geffner, H. 2014. Width-based Algorithms for Classical Planning: New Results. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 1059–1060. IOS Press.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In Singh, S.; and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3590–3596. AAAI Press.

Nakhost, H.; and Müller, M. 2009. Monte-Carlo Exploration for Deterministic Planning. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1766–1771. AAAI Press.

Richter, S.; and Helmert, M. 2009. Preferred Operators and Deferred Evaluation in Satisficing Planning. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 273–280. AAAI Press.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks Revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 975–982. AAAI Press.

Richter, S.; and Westphal, M. 2008. The LAMA Planner — Using Landmark Counting in Heuristic Search. IPC 2008 short papers, http://ipc.informatik.uni-freiburg.de/Planners.

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

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011 (planner abstract). In *IPC 2011 Planner Abstracts*, 50–54.

Röger, G.; and Helmert, M. 2010. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In Brafman, R.; Geffner, H.; Hoffmann, J.; and Kautz, H., eds., *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 246–249. AAAI Press.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated Cost Partitioning for Optimal Classical Planning. *Journal of Artificial Intelligence Research*, 67: 129–167.

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

Shleyfman, A.; Tuisov, A.; and Domshlak, C. 2016. Blind Search for Atari-Like Online Planning Revisited. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3251–3257. AAAI Press.

Tuisov, A.; and Katz, M. 2021. The Fewer the Merrier: Pruning Preferred Operators with Novelty. In Zhou, Z.-H., ed., *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*, 4190–4196. IJCAI.

Vidal, V. 2011. YAHSP2: Keep It Simple, Stupid. In *IPC 2011 Planner Abstracts*, 83–90.

Xie, F.; Müller, M.; Holte, R. C.; and Imai, T. 2014. Type-based Exploration with Multiple Search Queues for Satisficing Planning. In Brodley, C. E.; and Stone, P., eds., *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2395–2401. AAAI Press.