Consolidating LAMA with Best-First Width Search

Augusto B. Corrêa¹, Jendrik Seipp²

¹University of Basel, Switzerland
²Linköping 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).⁴ 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; Francês 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(f₆)) 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(f₆) 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(f₆), 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.

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 \( S = \langle S, s_I, G, \text{succ} \rangle \), where \( S \) is a set of states, \( s_I \in S \) is the initial state, \( G \) is the goal description, \( \text{succ} \) 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 \( \text{succ}(s) \subseteq S \) 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 \( s_i \in \text{succ}(s_{i-1}) \) for \( i \in \{1, \ldots, n\} \). Path \( \tau \) is an s-plan if \( s_0 = s \) and \( G \subseteq s_n \) and a plan for \( S \) 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 \( h : S \rightarrow \mathbb{R}_+ \cup \{\infty\} \). 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 \( h^{\text{add}} \) (Bonet and Geffner 2001), \( h^{\text{FF}} \) (Hoffmann and Nebel 2001), and \( h^{\text{LM}} \) (Richter, Helmer, 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 \( w(s) \) 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 Gelfner 2017; Francès et al. 2017, 2018). For BFWS, the novelty \( w(h_1,\ldots,h_n)(s) \) 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. Exploration 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: \([h^{\text{FF}}, h^{\text{LM}}, h^{\text{LM}}] \), 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 \( h_1 \) 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 Gelfner 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\((f_0)\), where the open-list is ordered by \( f_0 = \langle w(h^{\text{LM}}, h^{\text{FF}}), \text{pref}, h^{\text{LM}}, w(h^{\text{FF}}) \rangle \) where \( \text{pref} \) is an indicator function yielding 1 for states reached via a preferred operator. Simpler versions include BFWS\((f_1)\), where \( f_4 = \langle w(h^{\text{LM}}, h^{\text{FF}}), h^{\text{LM}}, h^{\text{FF}} \rangle \), and BFWS\((f_2)\), where \( f_2 = \langle w(h^{\text{FF}}), h^{\text{FF}} \rangle \).

A BFWS-based planner, BFWS-Pref, 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\((f_0)\), then both planners use exactly the same information (with the exception of the novelty measures): \( h^{\text{FF}}, h^{\text{LM}} \), 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\((f_0)\) as an additional open list in LAMA. We call this modification LAMA-W\((f_0)\).

Before we evaluate this new planner, we present the experimental setup used throughout the paper. We implemented LAMA-W\((f_0)\) and BFWS\((f_0)\), alongside the sim-
We started from LAMA-W($f_6$) and boiled the novelty-based open list down to obtain LAMA-W($f_2^{LM}$), which then outperformed both LAMA and BFWS($f_6$). But who says that

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

<table>
<thead>
<tr>
<th></th>
<th>LAMA</th>
<th>BFWS($f_6$)</th>
<th>LAMA-W($f_6$)</th>
<th>LAMA-W($f_2^{LM}$)</th>
<th>LAMA-W($w_{(LM)}$)</th>
<th>LAMA-W($w_{(LM)}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage</td>
<td>2081</td>
<td>2042</td>
<td>2029</td>
<td>2037</td>
<td>2047</td>
<td>2113</td>
</tr>
<tr>
<td>Expansions Score</td>
<td>1629.31</td>
<td>1687.50</td>
<td>1359.53</td>
<td>1363.68</td>
<td>1345.63</td>
<td>1643.34</td>
</tr>
<tr>
<td>Agile Score</td>
<td>1737.43</td>
<td>1581.45</td>
<td>1593.69</td>
<td>1600.41</td>
<td>1587.31</td>
<td>1751.11</td>
</tr>
</tbody>
</table>

---

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

---

**Consolidating the LAMA Open-Lists**

We started from LAMA-W($f_6$) and boiled the novelty-based open list down to obtain LAMA-W($f_2^{LM}$), which then outperformed both LAMA and BFWS($f_6$). But who says that
Table 2: Results for the consolidation of LAMA open-lists in LAMA-$W(f^{2\text{LM}}_h)$ (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 $f^{2\text{LM}}_h$ open-list.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>LAMA</th>
<th>NOLAN</th>
<th>Diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td>barman-sat14 (20)</td>
<td>19</td>
<td>20</td>
<td>+1</td>
</tr>
<tr>
<td>caldera-split-sat18 (20)</td>
<td>6</td>
<td>10</td>
<td>+4</td>
</tr>
<tr>
<td>cavediving-14 (20)</td>
<td>7</td>
<td>8</td>
<td>+1</td>
</tr>
<tr>
<td>data-network-sat18 (20)</td>
<td>11</td>
<td>15</td>
<td>+4</td>
</tr>
<tr>
<td>depot (22)</td>
<td>20</td>
<td>22</td>
<td>+2</td>
</tr>
<tr>
<td>freecell (80)</td>
<td>79</td>
<td>80</td>
<td>+1</td>
</tr>
<tr>
<td>logistics98 (35)</td>
<td>34</td>
<td>35</td>
<td>+1</td>
</tr>
<tr>
<td>maintenance-sat14 (20)</td>
<td>11</td>
<td>12</td>
<td>+1</td>
</tr>
<tr>
<td>nomystry-sat11 (20)</td>
<td>13</td>
<td>17</td>
<td>+4</td>
</tr>
<tr>
<td>nurikabe-sat18 (20)</td>
<td>16</td>
<td>17</td>
<td>+1</td>
</tr>
<tr>
<td>pathways (30)</td>
<td>23</td>
<td>25</td>
<td>+2</td>
</tr>
<tr>
<td>pipesworld-notankage (50)</td>
<td>44</td>
<td>45</td>
<td>+1</td>
</tr>
<tr>
<td>pipesworld-tankage (50)</td>
<td>41</td>
<td>42</td>
<td>+1</td>
</tr>
<tr>
<td>ricochet-robots-sat23 (20)</td>
<td>6</td>
<td>14</td>
<td>+8</td>
</tr>
<tr>
<td>rubiks-cube-sat23 (20)</td>
<td>20</td>
<td>17</td>
<td>-3</td>
</tr>
<tr>
<td>snake-sat18 (20)</td>
<td>4</td>
<td>9</td>
<td>+5</td>
</tr>
<tr>
<td>storage (30)</td>
<td>19</td>
<td>26</td>
<td>+7</td>
</tr>
<tr>
<td>tetris-sat14 (20)</td>
<td>17</td>
<td>20</td>
<td>+3</td>
</tr>
<tr>
<td>thoughtful-sat14 (20)</td>
<td>16</td>
<td>18</td>
<td>+2</td>
</tr>
<tr>
<td>tidybot-sat11 (20)</td>
<td>18</td>
<td>20</td>
<td>+2</td>
</tr>
<tr>
<td>transport-sat11 (20)</td>
<td>15</td>
<td>18</td>
<td>+3</td>
</tr>
<tr>
<td>transport-sat14 (20)</td>
<td>11</td>
<td>12</td>
<td>+1</td>
</tr>
<tr>
<td>trucks (30)</td>
<td>15</td>
<td>17</td>
<td>+2</td>
</tr>
<tr>
<td>trucks-strips (30)</td>
<td>14</td>
<td>16</td>
<td>+2</td>
</tr>
<tr>
<td>visitall-sat11 (20)</td>
<td>20</td>
<td>19</td>
<td>-1</td>
</tr>
<tr>
<td>visitall-sat14 (20)</td>
<td>20</td>
<td>4</td>
<td>-16</td>
</tr>
</tbody>
</table>

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-$W(f^{2\text{LM}}_h)$ by removing subsets of LAMA open-lists from LAMA-$W(f^{2\text{LM}}_h)$.

As explained above, the original LAMA planner uses four open-lists: $[h^{FF}, h^{FF}_+, h^{LM}, h^{LM}_+]$, while LAMA-$W(f^{2\text{LM}}_h)$ has five: $[h^{FF}, h^{FF}_+, h^{LM}, h^{LM}_+, f^{2\text{LM}}_h]$. Table 1 shows the effect of removing combinations of of the original 4 queues in LAMA from LAMA-$W(f^{2\text{LM}}_h)$. The two strongest versions use three open-lists from LAMA plus the $f^{2\text{LM}}_h$ open-list. In comparison to LAMA-$W(f^{2\text{LM}}_h)$, 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
We call it NOLAN because it uses (among other features) NOvelty, Landmarks and Alternative. The accumulated coverage over time by NOLAN is almost identical LAMA-W($f_{2}^{LM}$) 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 $h^{CFF}$ heuristic (Fickert and Hoffmann 2017), as the result from the lookahead can be used to trigger the refinement procedure of $h^{CFF}$. This idea was also used in the OLFF planner (Fickert and Hoffmann 2018) from the IPC 2018. We include OLFF 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 ideas 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 satisfying planner Mercury (Katz and Hoffmann 2014) with novelty heuristics (Katz et al. 2017); OLFF (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 OLFF planner.

Table 4 compares NOLAN with all state-of-the-art planners. 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($f_{h}$). 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 look-aheads, 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


