Adding Local Exploration to Greedy Best-First Search in Satisficing Planning

Fan Xie and Martin Müller and Robert Holte
Computing Science, University of Alberta
Edmonton, Canada
{fxie2,mmueller,robert.holte}@ualberta.ca

Abstract
Greedy Best-First Search (GBFS) is a powerful algorithm at the heart of many state of the art satisficing planners. One major weakness of GBFS is its behavior in so-called uninformative heuristic regions (UHRs) - parts of the search space in which no heuristic provides guidance towards states with improved heuristic values.

This work analyzes the problem of UHRs in planning in detail, and proposes a two level search framework as a solution. In Greedy Best-First Search with Local Exploration (GBFS-LE), a local exploration is started from within a global GBFS whenever the search seems stuck in UHRs.

Two different local exploration strategies are developed and evaluated experimentally: Local GBFS (LS) and Local Random Walk Search (LRW). The two new planners LAMA-LS and LAMA-LRW integrate these strategies into the GBFS component of LAMA-2011. Both are shown to yield clear improvements in terms of both coverage and search time on standard International Planning Competition benchmarks, especially for domains that are proven to have large or unbounded UHRs.

Introduction
In the latest International Planning Competition IPC-2011 (García-Olaya, Jiménez, and Linares López 2011), the planner LAMA-2011 (Richter and Westphal 2010) was the clear winner of the sequential satisficing track, by both measures of coverage and plan quality. LAMA-2011 finds a first solution using Greedy Best-First Search (GBFS) (Bonet and Geffner 2001; Helmert 2006) with popular enhancements such as Preferred Operators, Deferred Evaluation (Richter and Helmert 2009) and Multi-Heuristic search (Richter and Westphal 2010). Solutions are improved using restarting weighted A*.

GBFS always expands a node \( n \) that is closest to a goal state according to a heuristic \( h \). While GBFS makes no guarantees about solution quality, it can often find a solution quickly. The performance of GBFS strongly depends on the quality of \( h \). Misleading or uninformative heuristics can massively increase its running time.

The main focus of this paper is on one such problem with GBFS: uninformative heuristic regions (UHRs), which include local minima and plateaus. A local minimum is a state with minimum \( h \)-value within a local region which is not a global minimum. A plateau is an area of the state space where all states have the same heuristic value. GBFS, because of its open list, can get stuck in multiple UHRs at the same time.
Search Strategies for Escaping UHRs

There are several approaches to attack the UHR problem. Better quality heuristics (Hoffmann and Nebel 2001; Helmert 2004; Helmert and Geffner 2008) can shrink the size of UHRs, as can combining several heuristics (Richter and Westphal 2010; Röger and Helmert 2010). Additional knowledge from heuristic computation or from problem structure can be utilized in order to escape from UHRs. Examples are helpful actions (Hoffmann and Nebel 2001) and explorative probes (Lipovetzky and Geffner 2011). The third popular approach is to develop search algorithms that are less sensitive to flaws in heuristics. Algorithms which add a global exploration component to the search, which is especially important for escaping from unrecognized dead ends, include restarting (Nakhost and Müller 2009; Coles, Fox, and Smith 2007; Richter, Thayer, and Ruml 2010) and non-greedy node expansion (Valenzano et al. 2014; Imai and Kishimoto 2011; Xie et al. 2014). The current paper focuses on another direction: adding a local exploration component to the globally greedy GBFS algorithm.

The planner Marvin adds machine-learned plateau-escaping macro-actions to enforced hill-climbing (Coles and Smith 2007). YAHSP constructs macro actions from FF’s relaxed planning graph (Vidal 2004). IdentiTem adds exploration by expanding a sequence of actions chosen probabilistically, and proposes a framework for escaping from local minima in local-search forward-chaining planning (Coles, Fox, and Smith 2007). Arvand (Nakhost and Müller 2009) uses random walks to explore quickly and deeply. Arvand-LS (Xie, Nakhost, and Müller 2012) combines random walks with local greedy best-first search, while Roamer (Lu et al. 2011) adds exploration to LAMA-2008 by using fixed-length random walks. Analysis in (Nakhost and Müller 2012) shows that while random walks outperform GBFS in many plateau escape problems, they fail badly in domains such as Sokoban, where a precise action sequence must be found to escape. However, while escaping from UHRs has been well studied in the context of these local search based planners, there is comparatively little research on how to use search for escaping UHRs in the context of GBFS. This paper begins to fill this gap.

GBFS-LE: GBFS with Local Exploration

The new technique of Greedy Best-First Search with Local Exploration (GBFS-LE) uses local exploration whenever a global GBFS (G-GBFS) seems stuck. This approach is similar to GBFS-LR (Lu et al. 2011) which adds exploration to LAMA-2008 by using fixed-length random walks. GBFS-LE uses local random walks to explore quickly and deeply. GBFS-LE focuses on another direction: adding a local exploration component to the search, which is especially important for escaping from unrecognized dead ends.

The planner Marvin adds machine-learned plateau-escaping macro-actions to enforced hill-climbing (Coles and Smith 2007). YAHSP constructs macro actions from FF’s relaxed planning graph (Vidal 2004). IdentiTem adds exploration by expanding a sequence of actions chosen probabilistically, and proposes a framework for escaping from local minima in local-search forward-chaining planning (Coles, Fox, and Smith 2007). Arvand (Nakhost and Müller 2009) uses random walks to explore quickly and deeply. Arvand-LS (Xie, Nakhost, and Müller 2012) combines random walks with local greedy best-first search, while Roamer (Lu et al. 2011) adds exploration to LAMA-2008 by using fixed-length random walks. Analysis in (Nakhost and Müller 2012) shows that while random walks outperform GBFS in many plateau escape problems, they fail badly in domains such as Sokoban, where a precise action sequence must be found to escape. However, while escaping from UHRs has been well studied in the context of these local search based planners, there is comparatively little research on how to use search for escaping UHRs in the context of GBFS. This paper begins to fill this gap.

GBFS-LE: GBFS with Local Exploration

The new technique of Greedy Best-First Search with Local Exploration (GBFS-LE) uses local exploration whenever a global GBFS (G-GBFS) seems stuck. This approach is similar to GBFS-LR (Lu et al. 2011) which adds exploration to LAMA-2008 by using fixed-length random walks. GBFS-LE uses local random walks to explore quickly and deeply. GBFS-LE focuses on another direction: adding a local exploration component to the search, which is especially important for escaping from unrecognized dead ends.
Algorithm 1 GBFS-LE

Input Initial state $I$, goal states $G$
Parameter STALL_SIZE, MAX_LOCAL_TRY
Output A solution plan

1: $(open, h_{min}) \leftarrow ([I], h(I))$
2: stalled $\leftarrow$ 0; nuLocalTry $\leftarrow$ 0
3: while open $\neq \emptyset$ do
4: $n \leftarrow$ open.remove_min()
5: if $n \in G$ then
6: return plan from $I$ to $n$
7: end if
8: closed.insert($n$)
9: for each $v \in$ successors($n$) do
10: if $v \notin$ closed then
11: open.insert($v$, $h(v)$)
12: if $h_{min} > h(v)$ then
13: $h_{min} \leftarrow h(v)$
14: stalled $\leftarrow$ 0; nuLocalTry $\leftarrow$ 0
15: else
16: stalled $\leftarrow$ stalled + 1
17: end if
18: end if
19: end for
20: if stalled $\geq$ STALL_SIZE and nuLocalTry $< \text{MAX_LOCAL_TRY}$ then
21: $n \leftarrow$ open.remove_min()
22: LocalExplore($n$){local GBFS or random walks}
23: stalled $\leftarrow$ 0; nuLocalTry $\leftarrow$ nuLocalTry + 1
24: end if
25: end while

Local GBFS search starting from node $n$, LocalExplore($n$) = LS($n$), which shares the closed list of G-GBFS, but maintains its own separate open list local_open that is cleared before each local search. LS($n$) succeeds if it finds a node $v$ with $h(v) < h_{min}$ before it exceeds the LSIZE limit. In any case, the remaining nodes in local_open are merged into the global open list. A local search tree grown from a single node $n$ is much more focused and grows much more quickly than the global open list in G-GBFS. It also restricts the search to a single plateau, while G-GBFS can get stuck when exploring many separate plateaus simultaneously. Both G-GBFS and LS($n$) use a first-in-first-out tie-breaking rule, since last-in-first-out did not work well: it often led to long aimless walks within a UHR.

The second local exploration strategy tested is local random walk search, LocalExplore($n$) = LRW($n$). The implementation of random walks from the Arvand planner (Nakhost and Müller 2009; Nakhost et al. 2011) is used. LRW($n$) runs up to a pre-set number of random walks starting from node $n$, and evaluates the endpoint of each walk using $h^{FP}$. Like LS($n$), LRW($n$) succeeds if it finds a node $v$ with $h(v) < h_{min}$ within its exploration limit. In this case, $v$ is added to the global open list, and the path from $n$ to $v$ is stored for future plan extraction. In case of failure, unlike LS($n$), no information is kept.

Parameters, as in Arvand-2011, are expressed as a tuple $(\text{len_walk}, e\_rate, e\_period, \text{WalkType})$ (Nakhost and Müller 2009). Random walk length scaling is controlled by an initial walk length of $\text{len_walk}$, an extension rate of $e\_rate$ and an extension period of \text{NUMWALKS} * $e\_period$. This is very different from Roamer, which uses fixed length random walks. The choices for \text{WalkType} are pure random (\text{PURE}) and Monte Carlo Helpful Actions (MHA), which bias random walks by helpful actions. For example, in configuration (1, 2, 0.1, MHA) all random walks use the MHA walk type, and if $h_{min}$ does not improve for \text{NUMWALKS} * 0.1 random walks, then the length of walks, $\text{len_walk}$, which starts at 1, will be doubled. LRW was tested with the following two configurations: (1, 2, 0.1, MHA), which is used with preferred operators, and (1, 2, 0.1, \text{PURE}).

The example of Figure 2 is solved much faster, in around 1 second, by both GBFS-LS and GBFS-LRW, while GBFS needs 771 seconds. The three algorithms built exactly the same search trees when they first achieved the minimum $h$-value 6. The local GBFS in GBFS-LS, because of focusing on one branch, found a 5 step path that decreases the minimum $h$-value using only 10 expansions. The $h$-values along the path were 6, 7, 6 and 4, showing an initial increase before decreasing. $h$-values along GBFS-LRW’s path also increased before decreasing. In contrast, GBFS gets stuck in multiple separate $h$-plateaus since it needs to expand over 10000 nodes with $h$-value 6, which were distributed in many different parts of the search tree. Only after exhausting these, it expands the first node with $h = 7$. In this example, the local explorations, which expand or visit higher $h$-value nodes earlier, massively speed up the escape from UHRs.

There are several major differences between GBFS-LS and GBFS-LRW. GBFS-LS keeps all the information gathered during local searches by copying its nodes into the global open list at the end. GBFS-LRW keeps only endpoints that improve $h_{min}$ and the paths leading to them. This causes a difference in how often the local search should be called. For GBFS-LS, it is generally safe to do more local searches, while over-use of local search in GBFS-LRW can waste search effort. This suggests using more conservative settings for the parameters \text{MAX_LOCAL_TRY} and LSIZE in LRW($n$). The two algorithms also explore UHRs very differently. LS($n$) systematically searches the subtree of $n$, while LRW($n$) samples paths leading from $n$ sparsely but deeply.

**Experimental Results**

Experiments were run on a set of 2112 problems in 54 domains from the seven International Planning Competitions which are publicly available, using one core of a 2.8 GHz machine with 4 GB memory and 30 minutes per instance. Results for planners which use randomization are averaged over five runs. All planners are implemented on the Fast
Downward code base FD-2011 (Helmert 2006). The translation from PDDL to SAS+ was done only once, and this common preprocessing time is not counted in the 30 minutes. Parameters were set as follows: \( \text{STALL\_SIZE} = 1000 \) for both algorithms. \( (\text{MAX\_LOCAL\_TRY}, \text{LSIZE}) = (100, 100) \) for GBFS-LS and \( (10, 100) \) for GBFS-LRW.

Local Search Topology for \( h^+ \)

For the purpose of experiments on UHR, the detailed classification by \( h^+ \) of Figure 1 can be coarsened into three broad categories:

- **Unrecognized-Deadend**: 195 problems from 4 domains with unrecognized dead ends: Mystery, Mprime, Freecell and Airport.
- **Large-UHR**: 383 problems from domains with UHRs which are large or of unbounded exit distance, but with recognized dead ends; column 3 in Figure 1, plus the top two rows of columns 1 and 2.
- **Small-UHR**: 669 problems from domains without UHRs, or with only small UHRs, corresponding to columns 1 and 2 in the bottom row of Figure 1.

Note, problems from these three categories are only a subset of the total 2112 problems. Only part of the 54 domains were analyzed in Hoffmann’s work (Hoffmann 2011).

Performance of Baseline Algorithms

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>GBFS</th>
<th>GBFS-LS</th>
<th>GBFS-LRW</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF</td>
<td>1561</td>
<td>1657</td>
<td>1619.4</td>
</tr>
<tr>
<td>CG</td>
<td>1513</td>
<td>1602</td>
<td>1573.2</td>
</tr>
<tr>
<td>CEA</td>
<td>1498</td>
<td>1603</td>
<td>1615.2</td>
</tr>
</tbody>
</table>

Table 1: IPC coverage out of 2112 for GBFS with and without local exploration, and three standard heuristics.

The baseline study evaluates GBFS, GBFS-LS and GBFS-LRW without the common planning enhancements of preferred operators, deferred evaluation and multi-heuristics. Three widely used planning heuristics are tested: FF (Hoffmann and Nebel 2001), causal graph (CG) (Helmert 2004) and context-enhanced additive (CEA) (Helmert and Geffner 2008). Table 1 shows the coverage on all 2112 IPC instances. Both GBFS-LS and GBFS-LRW outperform GBFS by a substantial margin for all 3 heuristics.

Figure 3(a) compares the time usage of the two proposed algorithms with GBFS using \( h^{FP} \) over all IPC benchmarks. Every point in the figure represents one instance, plotting the search time for GBFS on the \( x \)-axis against GBFS-LS (top) and GBFS-LRW (bottom) on the \( y \)-axis. Only problems for which both algorithms need at least 0.1 seconds are shown. Points below the main diagonal represent instances that the new algorithms solve faster than GBFS. For ease of comparison, additional reference lines indicate \( 2 \times \), \( 10 \times \) and \( 50 \times \) relative speed. Data points within a factor of 2 are shown in grey in order to highlight the instances with substantial differences. Problems that were only solved by one algorithm within the 1800 second time limit are included at \( x = 10000 \) or \( y = 10000 \). Both new algorithms show substantial improvements in search time over GBFS.

Figures 3(b) and (c) restrict the comparison to Large-UHR and Small-UHR respectively. In Large-UHR, GBFS-LS and GBFS-LRW solve 19 (+9.7%) and 30 (+15.3%) more problems than GBFS (195/383) respectively. Both outperform GBFS in search time. However, in Small-UHR, GBFS-LS and GBFS-LRW only solve 3 (+0.5%) and 7 (+1.1%) more problems than GBFS (634/669), and there is very little difference in search time among the three algorithms. This result clearly illustrates the relationship between the size of UHRs and the performance of the two local exploration techniques. For Unrecognized-Deadend, GBFS-LS is slightly slower than GBFS with the same coverage (162/195), while GBFS-LRW is slightly faster and solves 7 (+3.7%) more problems. The effect of local exploration on the performance in the case of unrecognized dead-ends is not clear-cut.

Performance with Search Enhancements

Experiments in this section test the two proposed algorithms when three common planning enhancements are added: Deferred Evaluation, Preferred Operators and Multiple Heuristics. \( h^{FF} \) is used as the primary heuristic in all cases.

- **Deferred Evaluation** delays state evaluation and uses the parent’s heuristic value in the priority queue (Richter and Helmert 2009). This technique is used in G-GBFS and \( LS(n) \), but not in the endpoint-only evaluation of random walks in \( LRW(n) \).

- **The Preferred Operators** enhancement keeps states reached via a preferred operator, such as helpful actions in \( h^{FP} \), in an additional open list (Richter and Helmert 2009). An extra preferred open list is also added to \( LS(n) \). Boosting with default parameter 1000 is used, and Preferred Operator first ordering is used for tie-breaking as in LAMA-2011 (Richter and Westphal 2010). In \( LRW(n) \), preferred operators are used in form of the Monte Carlo with Helpful Actions (MHA) technique (Nakhost and Müller 2009), which biases random walks towards using operators which are often preferred.

- **The Multi-Heuristics** approach maintains additional open lists in which states are evaluated by other heuristic functions. Because of its proven strong performance in LAMA, the Landmark count heuristic \( h^{lm} \) (Richter, Helmert, and Westphal 2008) is used as the second heuristic. Both G-GBFS and \( LS(n) \) use a round robin strategy for picking the next node to expand. In Fast Downward, \( h^{lm} \) is calculated incrementally from the parent node. When Multi-Heuristics is applied to GBFS-LRW, the \( LRW(n) \) part still uses \( h^{FP} \) because the path-dependent landmark computation was not implemented for random walks. When \( LRW(n) \) finds an heuristically improved state \( s \), GBFS-LRW evaluates and expands all states along the path to \( s \) in order to allow the path-dependent computation of \( h^{lm}(s) \) in G-GBFS. Without Multi-Heuristics, only \( s \) itself is inserted to the open list.

Table 2 shows the experimental results on all IPC domains. Used as a single enhancement, Preferred Operators
improves all three algorithms. Deferred Evaluation improves GBFS-LS and GBFS-LRW, but fails for GBFS, mainly due to plateaus caused by the less informative node evaluation (Richter and Helmert 2009). In GBFS-LS and GBFS-LRW, the benefit of faster search outweighs the weaker evaluation. Multi-Heuristics strongly improves GBFS and GBFS-LS, but is only a modest success in GBFS-LRW. This is not surprising since LRW(n) does not use h^lm, and in order to evaluate the new best states generated by LRW(n) with h^lm in G-GBFS, all nodes on the random walk path need to be evaluated, which degrades performance.

Table 2: Number of instances solved with search enhancements, out of 2112. PO = Preferred Operators, DE = Deferred Evaluation, MH = Multi-Heuristic.

<table>
<thead>
<tr>
<th>Enhancement</th>
<th>GBFS</th>
<th>GBFS-LS</th>
<th>GBFS-LRW</th>
</tr>
</thead>
<tbody>
<tr>
<td>(none)</td>
<td>1561</td>
<td>1657</td>
<td>1619.4</td>
</tr>
<tr>
<td>PO</td>
<td>1826</td>
<td>1851</td>
<td>1827.4</td>
</tr>
<tr>
<td>DE</td>
<td>1535</td>
<td>1721</td>
<td>1635</td>
</tr>
<tr>
<td>MH</td>
<td>1851</td>
<td>1874</td>
<td>1688.4</td>
</tr>
<tr>
<td>PO + DE</td>
<td>1871</td>
<td>1889</td>
<td>1880.6</td>
</tr>
<tr>
<td>PO + MH</td>
<td>1850</td>
<td>1874</td>
<td>1854.2</td>
</tr>
<tr>
<td>DE + MH</td>
<td>1660</td>
<td>1764</td>
<td>1730.2</td>
</tr>
<tr>
<td>PO + DE + MH</td>
<td>1913</td>
<td>1931</td>
<td>1925.4</td>
</tr>
</tbody>
</table>

Table 2: Number of instances solved with search enhancements, out of 2112. PO = Preferred Operators, DE = Deferred Evaluation, MH = Multi-Heuristic.

Comparing State of the Art Planners in terms of Coverage and Search Time

The final row in Table 2 shows coverage results when all three enhancements are applied. The performance comparisons in this section use this best known configuration in terms of coverage for three algorithms based on GBFS, GBFS-LS and GBFS-LRW, which closely correspond to the “coverage-only” first phase of the LAMA-2011 planner:

- **LAMA-2011**: only the first GBFS iteration of LAMA is run, with deferred evaluation, preferred operators and multi-heuristics with h^FF and h^lm (Richter and Westphal 2010).
- **LAMA-LS**: Configured like LAMA-2011, but with GBFS replaced by GBFS-LS.
- **LAMA-LRW**: GBFS in LAMA-2011 is replaced by GBFS-LRW.

Table 3 shows the coverage results per domain. LAMA-LS has the best overall coverage. 18 more than LAMA-2011, closely followed by LAMA-LRW. LAMA-LS solves more problems in 7 of the 10 domains where LAMA and LAMA-LS differ in coverage. This number for LAMA-LRW is 7 out of 11. Although LAMA-LRW uses a randomized algorithm, our 5 runs for LAMA-LRW had quite stable results: 1927, 1924, 1926, 1924 and 1926. By comparison, adding the landmark count heuristic, which differentiates LAMA-2011 from other planners based on the Fast Downward code base, improves the coverage of LAMA-2011 by 42, from 1871 to 1913.

Using the same format as Figure 3 for baseline GBFS, Figure 4 compares the search time of the three planners over the IPC benchmark. Both LAMA-LS and LAMA-LRW show a clear overall improvement over LAMA-2011 in terms of speed. The benefit of local exploration for search time in Large-UHR still holds even with all enhancements on. Both LAMA-LS and LAMA-LRW solve 12 more problems (4.1%) than LAMA-2011’s 290/383 in Large-UHR, while in Small-UHR they solve 1 and 2 fewer problems re-

Figure 3: Comparison of time usage of the three baseline algorithms. 10000 corresponds to runs that timed out or ran out of memory. Results shown for one typical run of GBFS-LRW, which is selected by comparing all 5 runs and picking the most typical one. They are all very similar.
Figure 4: Comparison of search time of LAMA-2011 with LAMA-LS and LAMA-LRW. A typical single run is used for LAMA-LRW.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Size</th>
<th>LAMA-2011</th>
<th>LAMA-LS</th>
<th>LAMA-LRW</th>
</tr>
</thead>
<tbody>
<tr>
<td>00-miconic-ful</td>
<td>150</td>
<td>136</td>
<td>136</td>
<td>135.6</td>
</tr>
<tr>
<td>02-depot</td>
<td>22</td>
<td>20</td>
<td>20</td>
<td>19.6</td>
</tr>
<tr>
<td>02-freecell</td>
<td>80</td>
<td>78</td>
<td>79</td>
<td>78.2</td>
</tr>
<tr>
<td>04-airport-str</td>
<td>50</td>
<td>32</td>
<td>34</td>
<td>32.8</td>
</tr>
<tr>
<td>04-notankage</td>
<td>50</td>
<td>44</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>04-optical-tel</td>
<td>48</td>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>04-philosoph</td>
<td>48</td>
<td>39</td>
<td>47</td>
<td>47.8</td>
</tr>
<tr>
<td>04-satellite</td>
<td>36</td>
<td>36</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>06-storage</td>
<td>30</td>
<td>18</td>
<td>23</td>
<td>21</td>
</tr>
<tr>
<td>06-tankage</td>
<td>50</td>
<td>41</td>
<td>41</td>
<td>42</td>
</tr>
<tr>
<td>08-transport</td>
<td>30</td>
<td>29</td>
<td>30</td>
<td>29.6</td>
</tr>
<tr>
<td>11-floortile</td>
<td>20</td>
<td>6</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>11-parking</td>
<td>20</td>
<td>18</td>
<td>20</td>
<td>16.8</td>
</tr>
<tr>
<td>11-transport</td>
<td>20</td>
<td>16</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>Total</td>
<td>2112</td>
<td>1913</td>
<td>1931</td>
<td>1925.4</td>
</tr>
<tr>
<td>Unsolved</td>
<td>199</td>
<td>181</td>
<td>186.6</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Domains with different coverage for the three planners. 33 domains with 100% coverage and 7 further domains with identical coverage for all planners are not shown.

Conclusions and Future Work

While local exploration has been investigated before in the context of local search planners, it also serves to facilitate escaping from UHRs for greedy best-first search. The new framework of GBFS-LE, GBFS with Local Exploration, has been tested successfully in two different realizations, adding local greedy best-first search in GBFS-LS and random walks in GBFS-LRW.

Future work should explore more types of local search such as FF’s enforced hill-climbing (Hoffmann and Nebel 2001), and try to combine different local exploration methods in a principled way. One open problem of GBFS-LE is that it does not have a mechanism for dealing with unrecognized dead-ends. Local exploration in GBFS-LE always starts from the heuristically most promising state in the global open list, which might be mostly filled with nodes from such dead-ends. In domains such as 2011-nomystery (Nakhost, Hoffmann, and Müller 2012), almost all exploration will occur within such dead ends and therefore be useless. It would be interesting to combine GBFS-LE with an algorithm for increased global-level exploration, such as DBFS (Imai and Kishimoto 2011) and Type-GBFS (Xie et al. 2014).

Acknowledgements

The authors wish to thank the anonymous referees for their valuable advice. This research was supported by GRAND NCE, a Canadian federally funded Network of Centres of Excellence, NSERC, the Natural Sciences and Engineering Research Council of Canada, and AITF, Alberta Innovates Technology Futures.
References


