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,rholte}@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 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 unbounded or large UHRs.[1]

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.

[1]Another version of this paper is published in AAAI 2014 (Xie, Müller, and Holte 2014)

The main focus of this paper is on one such problem with GBFS: uninformative heuristic regions (UHRs), which includes *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.

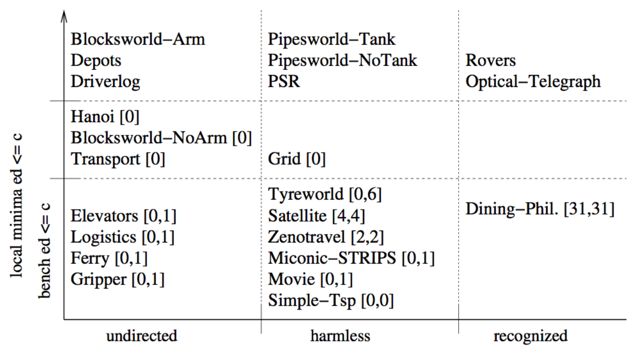


Figure 1: Overview of h^+ topology (Hoffmann 2011). Domains with unrecognized dead ends are not shown.

Hoffmann has studied the problem of UHRs for the case of the optimal relaxation heuristic h^+ (Hoffmann 2005; 2011). He classified a large number of planning benchmarks, shown in Figure 1, according to their *maximum exit distance* from plateaus and local minima, and by whether dead ends exist and are recognized by h^+. The current work proposes local exploration to improve GBFS. The focus of the analysis is on domains with a large or even unbounded maximum exit distance for plateaus and local minima, but without unrecognized dead ends. In these domains, there exists a plan from each state in an UHR (with $h^+ < \infty$).

As an example, the IPC domain 2004-notankage has no dead ends, but contains unbounded plateaus and local minima (Hoffmann 2011). Instance #21 shown in Figure 2 serves to illustrate a case of bad search behavior in GBFS due to UHRs. The figure plots the current minimum heuristic value h_{min} in the closed list on the x-axis against the log-scale cumulative search time needed to first reach h_{min}. The solid line is for GBFS with h^{FF}. The two huge increases

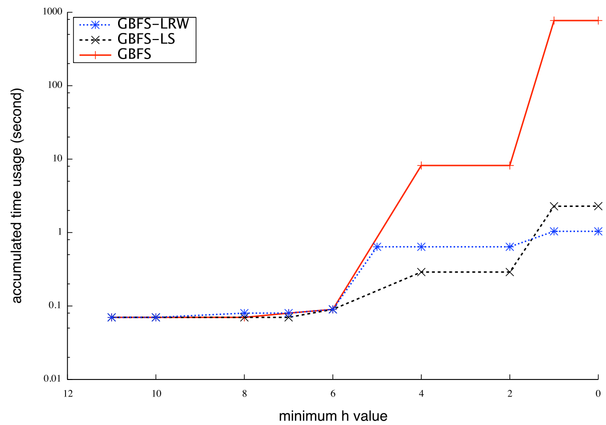


Figure 2: Cumulative search time (in seconds) of GBFS, GBFS-LS and GBFS-LRW with h^{FF} for first reaching a given h_{min} in 2004-notankage #21.

in search time, with the largest (763 seconds) for the step from $h_{min} = 2$ to $h_{min} = 1$, correspond to times when the search is stalled in multiple UHRs. Since the large majority of overall search time is used to inefficiently find an escape from UHRs, it seems natural to try switching to a secondary search strategy which is better at escaping. Such ideas have been tried several times before. This related work is reviewed and compared in the next section.

The current paper introduces a framework which adds a local search algorithm to GBFS in order to improve its behavior in UHRs. Two such algorithms, *local GBFS (LS(n))* and *local random walks (LRW(n))*, are designed to find quicker escapes from UHRs, starting from a node n within the UHRs. The main contributions of this work are:

- An analysis of the problem of UHRs in GBFS, and its consequences for limiting the performance of GBFS in current benchmark problems in satisficing planning.
- A new search framework, Greedy Best-First Search with Local Exploration (GBFS-LE), which runs a separate local search whenever the main global GBFS seems to be stuck. Two concrete local search algorithms, local GBFS (*LS*) and local random walks (*LRW*), are shown to be less sensitive to UHRs and when incorporated into GBFS are shown to outperform the baseline by a substantial margin over the IPC benchmarks.
- An analysis of the connection between Hoffmann's theoretical results on local search topology (Hoffmann 2005; 2011) and the performance of adding local exploration into GBFS.

The remainder of the paper is organized as follows: after a brief review of previous work on strategies for escaping from UHR, the new search framework GBFS-LE is introduced, compared with related work, and evaluated experimentally on IPC domains. A discussion of possible future work includes perspectives for addressing the early mistakes problem within GBFS-LE.

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) 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). Identidem 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. Nakhost and Müller's analysis (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. If G-GBFS fails to improve its minimum heuristic value h_{min} for a fixed number of node expansions, then GBFS-LE runs a small local search for exploration, *LocalExplore(n)*, from the best node n in a global-level open list. Algorithm 1 shows GBFS-LE. *STALL_SIZE* and *MAX_LOCAL_TRY*, used at Line 24, are parameters which control the tradeoff between global search and local exploration.

The main change from GBFS is the call to *LocalExplore(n)* at Line 26 whenever there has been no improvement in heuristic value over the last *STALL_SIZE* node expansions.

Two local exploration strategies were tested. The first is

Algorithm 1 GBFS-LE

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

```
if h(I) < ∞ then
    (open, h_min) ← ([I], h(I))
end if
stalled ← 0; nuLocalTry ← 0
while open ≠ ∅ do
    n ← open.remove_min(){FIFO tie-breaking}
    if n ∈ G then
        return plan from I to n
    end if
    closed.insert(n)
    for each v ∈ successors(n) do
        if v ∉ closed then
            if h(v) < ∞ then
                open.insert(v, h(v))
                if h_min > h(v) then
                    h_min ← h(v)
                    stalled ← 0; nuLocalTry ← 0
                else
                    stalled ← stalled + 1
                end if
            end if
        end if
    end for
    if stalled = STALL_SIZE
    and nuLocalTry < MAX_LOCAL_TRY then
        n ← open.peek_min()
        LocalExplore(n){local GBFS or random walks}
        stalled ← 0; nuLocalTry ← nuLocalTry + 1
    end if
end while
```

Algorithm 2 $LS(n)$, *local GBFS*

Input state n, goal states G, h_{min}{global variable}, $open$, $closed$
Parameter LSSIZE

```
1:  local_open ← [n]
2:  h_improved ← false
3:  for i = 1 to LSSIZE do
4:      if local_open = ∅ then
5:          return
6:      end if
7:      n ← local_open.remove_min() {FIFO
        tie-breaking}
8:      if n ∈ G then
9:          return plan from I to n
10:     end if
11:     closed.insert(n)
12:     for each v ∈ successors(n) do
13:         if v ∉ closed then
14:             if h(v) < ∞ then
15:                 local_open.insert(v, h(v))
16:                 if h_min > h(v) then
17:                     h_min ← h(v)
18:                     h_improved ← true
19:                 end if
20:             end if
21:         end if
22:     end for
23:     if h_improved then
24:         break
25:     end if
26: end for
27: merge(open,local_open)
28: return
```

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. $LS(n)$, as shown in Algorithm 2, succeeds if it finds a node v with $h(v) < h_{min}$ at Line 16 before it exceeds the *LSSIZE* 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 deep 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)*, as shown in Algorithm 3. 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 only the endpoint of each walk using h^{FF}. All intermediate states are checked for whether they are goal states. Like $LS(n)$, $LRW(n)$ succeeds if it finds a node v with $h(v) < h_{min}$ within its exploration limit at Line 15. 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 (len_walk, e_rate, e_period, $WalkType$) (Nakhost and Müller 2009). Random walk length scaling is controlled by an initial walk length of len_walk, an extension rate of e_rate and an extension period of $NUMWALKS * e\_period$. This is very different from Roamer, which uses fixed length random walks. $WalkType$ defines two different strategies for action selecting at Line 8: Monte Carlo Helpful Actions (*MHA*), which bias random walks by helpful actions, and pure random (*PURE*). 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 $NUMWALKS * 0.1$ random walks, then the length of walks, 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 $(1, 2, 0.1, PURE)$.

The example of Figure 2 is solved much faster, in around

Algorithm 3 *LRW(n), local random walk*

Input state n, goal states G, h_{min} {global variable}, *open*
Parameter LSSIZE

```
for i = 1 to LSSIZE do
    s ← n
    for j = 1 to LENGTH_WALK do
        A ← ApplicableActions(s)
        if A = ∅ then
            break
        end if
        a ← SelectAnActionFrom(A)
        s ← apply(s, a)
        if s ∈ G then
            open.insert(s, h(s))
            return
        end if
    end for
    if h(s) < h_min then
        open.insert(s, h(s))
        break
    end if
end for
return
```

1 second, by both GBFS-LS and GBFS-LRW, while GBFS needs 771 seconds. The three algorithms built exactly the same search trees until they first achieved the minimum h-value 6. The local GBFS in GBFS-LS, because it could focus 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, 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 search, while over-use of local search in GBFS-LRW can waste search effort[2]. This suggests using more conservative settings for the parameters *MAX_LOCAL_TRY* and *LSSIZE* 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.

[2] Each step in a random walk generates all children and randomly picks one, which is only slightly cheaper than one expansion by LS when Deferred Evaluation is applied.

Experimental Results

Experiments were run on a set of 2112 problems in 54 domains from the seven International Planning Competitions which are publicly available[3], 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: *STALL_SIZE = 1000* for both algorithms. (*MAX_LOCAL_TRY*, *LSSIZE*) = (100, 1000) for GBFS-LS and (10, 100) for GBFS-LRW.

Local Search Topology for h^+

For the purpose of experiments on UHRs, 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 a part of the 54 domains were analyzed by Hoffmann (2011).

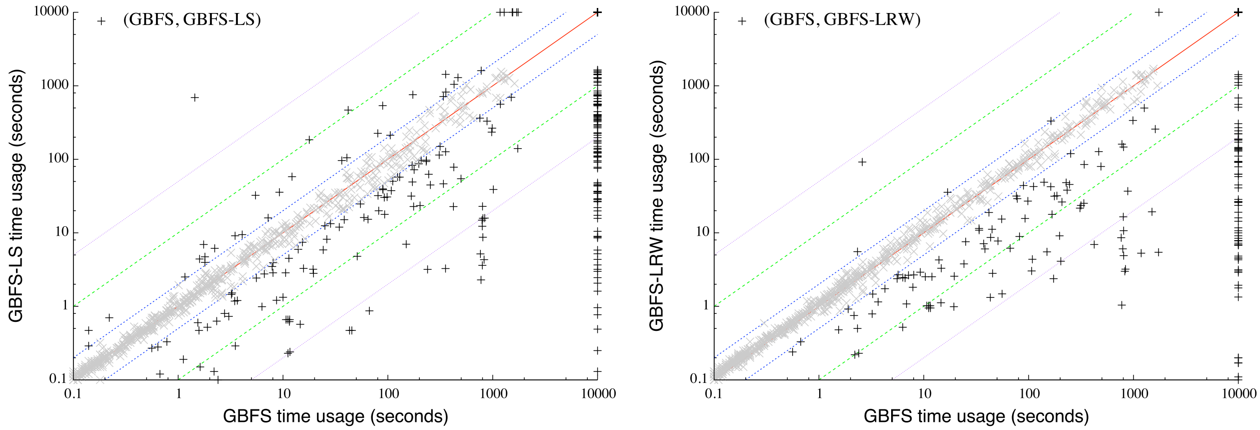
Performance of Baseline Algorithms

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). We use the distance-base versions for the three heuristics. They estimate the length of a solution path starting from the evaluated state. 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.

Heuristic	GBFS	GBFS-LS	GBFS-LRW
FF	1561	**1657**	1619.4
CG	1513	**1602**	1573.2
CEA	1498	1603	**1615.2**

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

[3] Our IPC test set does not include Hanoi, Ferry and Simple-Tsp from Figure 1.



(a) GBFS(X) vs GBFS-LS(Y)

(b) GBFS(X) vs GBFS-LRW(Y)

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.

Benchmarks	GBFS	GBFS-LS	GBFS-LRW
UR-Deadend(195)	162	162(0.0%)	**169(3.7%)**
Large-UHR(383)	195	214(9.7 %)	**225(15.3%)**
Small-UHR(669)	634	637 (0.5%)	**641(1.1%)**

Table 2: Coverage comparison on the three domain categories for GBFS and GBFS-LE with h^{FF}. UR-Deadend is short for Unrecognized-Deadend. The same typical run in Figure 3 is used for GBFS-LRW. Numbers in parentheses show coverage improvements compared to GBFS.

Figure 3 compares the time usage of the two proposed algorithms with GBFS using h^{FF} 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 (left) and GBFS-LRW (right) 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.

Figure 4 restricts the comparison to Unrecognized-Deadend, Large-UHR and Small-UHR respectively. Table 2 shows the overall coverages. 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^{FF}, 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^{FF} because the path-dependent landmark

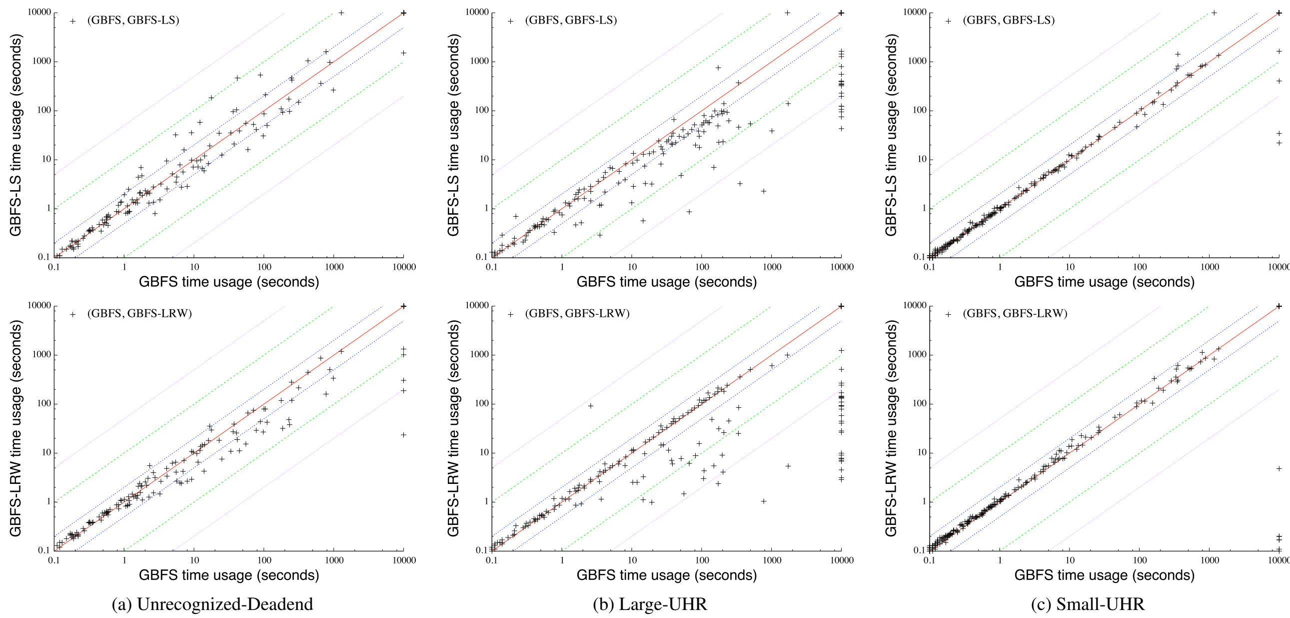


Figure 4: Comparison of time usage of the three baseline algorithms over the three different categories. 10000 corresponds to runs that timed out or ran out of memory. Results shown for one typical run of GBFS-LRW, which is selected by comparing all 5 runs and picking the most typical one. They are all very similar.

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 3 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. When combining two enhancements, all three algorithms achieve their best performance with Preferred Operators plus Deferred Evaluation. Figure 5 compares the time usage of the three algorithms in this case.

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

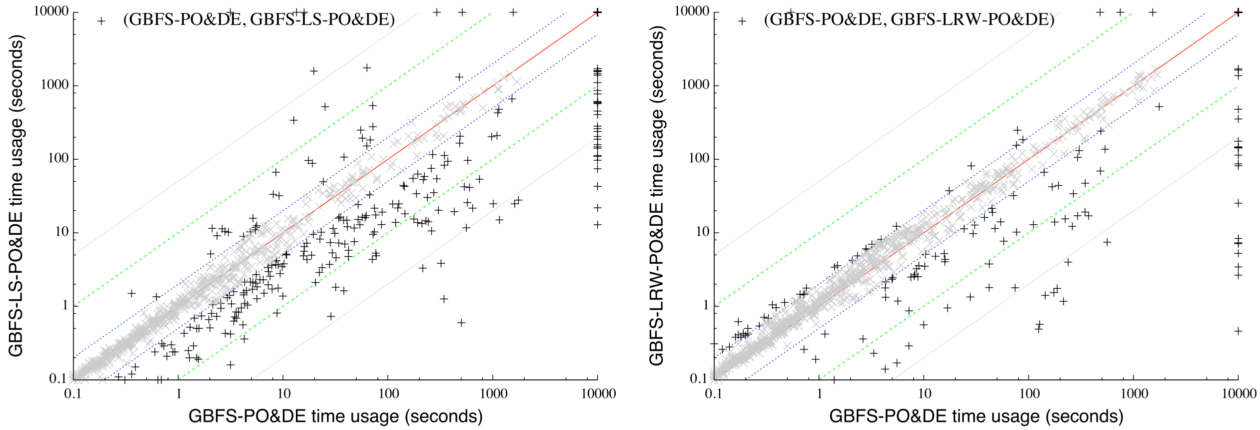
The final row in Table 3 shows coverage results when all three enhancements are applied. The performance comparisons in this section use this best known configuration in terms of coverage for three algorithms based on GBFS, GBFS-LS and GBFS-LRW, which closely correspond to the "coverage-only" first phase of the LAMA-2011 planner:

Enhancement	GBFS	GBFS-LS	GBFS-LRW
(none)	1561	1657	1619.4
PO	1826	1851	1827.4
DE	1535	1721	1635
MH	1851	1874	1688.4
PO + DE	1871	1889	1880.6
PO + MH	1850	1874	1854.2
DE + MH	1660	1764	1730.2
PO + DE + MH	**1913**	**1931**	**1925.4**

Table 3: Number of instances solved with search enhancements, out of 2112. PO = Preferred Operators, DE = Deferred Evaluation, MH = Multi-Heuristic.

- **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.

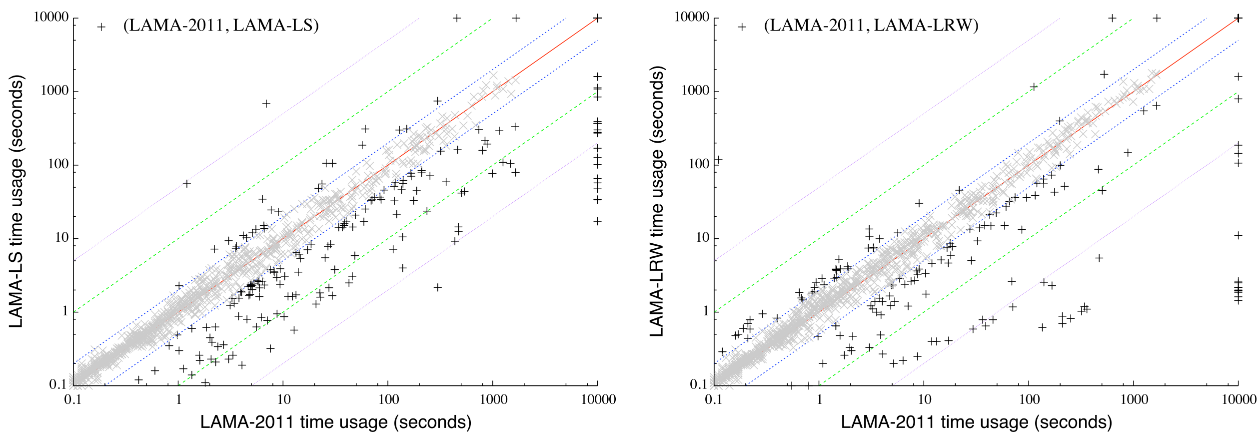
Table 4 shows the coverage results per domain. LAMA-LS has the best overall coverage, 18 more than LAMA-2011, closely followed by LAMA-LRW. LAMA-LS solves more problems in 7 of the 10 domains where LAMA and LAMA-LS differ in coverage. This number for LAMA-LRW is 7 out of 11. Although LAMA-LRW uses a 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-



GBFS-PO&DE(x) vs GBFS-LS-PO&DE(y)

GBFS-PO&DE(x) vs GBFS-LRW-PO&DE(y)

Figure 5: Comparison of time usage of the three baseline algorithms with Preferred Operators and Deferred Evaluation. 10000 corresponds to runs that timed out or ran out of memory. A typical single run of GBFS-LRW-PO&DE is shown.



(a) LAMA-2011(X) VS LAMA-LS(Y)

(b) LAMA-2011(X) VS LAMA-LRW(Y)

Figure 6: Comparison of time usage of LAMA-2011 with LAMA-LS and LAMA-LRW. A typical single run is used for LAMA-LRW.

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 6 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. In Figure 7, 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 respectively than LAMA-2011's 646/669. Table 5 compares coverages of the three planners over different categories.

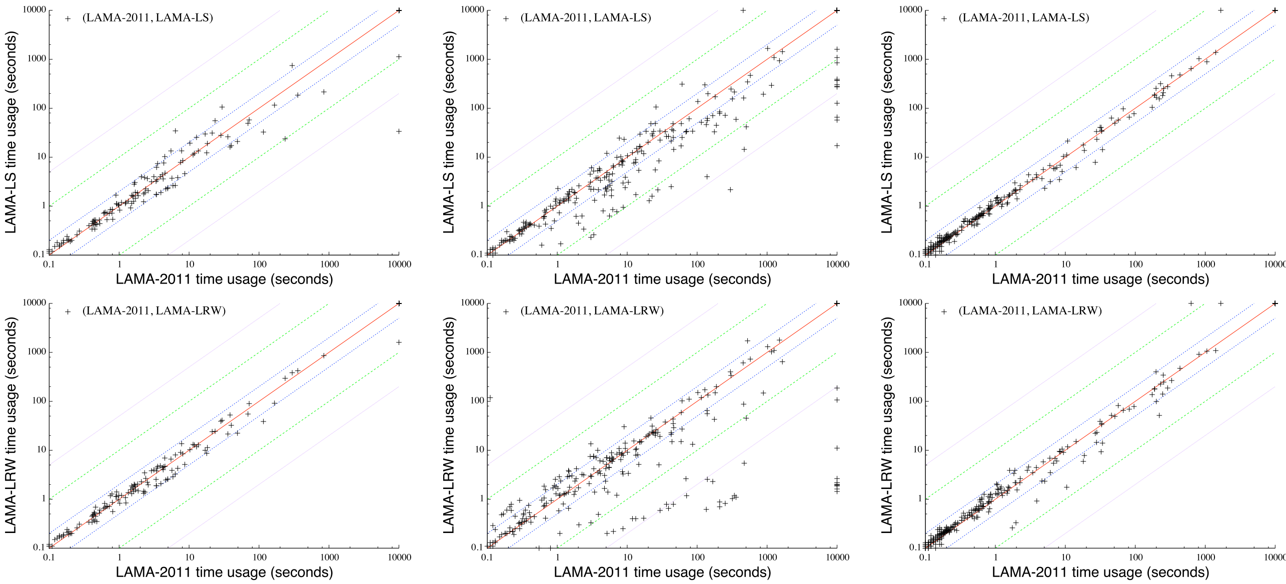
For further comparison, the coverage results of some other strong planners from IPC-2011 on the same hardware are: FDSS-2 solves 1912/2112, Arvand 1878.4/2112, Lama-2008 1809/2112, fd-auto-tune-2 1747/2112, and Probe 1706/1968 (failed on the ":derive" keyword in 144 problems).

Although the local explorations are inclined to increase the solution length, the influence is not clear-cut since they also solve more problems. The IPC-2011 style plan quality scores for LAMA-2011, LAMA-LS and LAMA-LRW are 1898.0, 1899.6 and 1900.5.

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 meth-



(a) Unrecognized-Deadend (b) Large-UHR (c) Small-UHR

Figure 7: Comparison of time usage of LAMA-2011 with LAMA-LS and LAMA-LRW over the three different categories. A typical single run is used for LAMA-LRW.

Domain	Size	LAMA-2011	LAMA-LS	LAMA-LRW
00-miconic-ful	150	**136**	**136**	135.6
02-depot	22	**20**	**20**	19.6
02-freecell	80	78	**79**	78 .2
04-airport-str	50	32	**34**	32.8
04-notankage	50	**44**	43	**44**
04-optical-tel	48	4	**6**	4
04-philosoph	48	39	47	**47.8**
04-satellite	36	**36**	35	35
06-storage	30	18	**23**	21
06-tankage	50	41	41	**42**
08-transport	30	29	**30**	**29.6**
11-floortile	20	**6**	5	**6**
11-parking	20	18	**20**	16.8
11-transport	20	16	16	**17**
All others	1396	1396	1396	1396
Total	2112	1913	1931	1925.4
Unsolved		199	181	186.6

Table 4: 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.

Benchmarks	LAMA-2011	LAMA-LS	LAMA-LRW
UR-Deadend(195)	164	**166(1.2%)**	165(0.6%)
Large-UHR(383)	290	**302(4.1 %)**	**302(4.1%)**
Small-UHR(669)	**646**	645 (-0.2%)	641(-0.3%)

Table 5: Coverages over the three domain categories for LAMA-2011, LAMA-LS and LAMA-LRW. UR-Deadend is short for Unrecognized-Deadend. The same typical run in Figure 6 is used for LAMA-LRW. Numbers in parentheses show coverage improvements compared to LAMA-2011.

ods 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

Bonet, B., and Geffner, H. 2001. Heuristic search planner 2.0. *AI Magazine* 22(3):77–80.

Coles, A., and Smith, A. 2007. Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research* 28:119–156.

Coles, A.; Fox, M.; and Smith, A. 2007. A new local-search algorithm for forward-chaining planning. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-2007)*, 89–96.

García-Olaya, A.; Jiménez, S.; and Linares López, C., eds. 2011. *The 2011 International Planning Competition*.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E. A., eds., *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS-2008)*, 140–147.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-2004)*, 161–170.

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

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

Hoffmann, J. 2005. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

Hoffmann, J. 2011. Where ignoring delete lists works, part II: Causal graphs. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-2011)*, 98–105.

Imai, T., and Kishimoto, A. 2011. A novel technique for avoiding plateaus of greedy best-first search in satisficing planning. In Burgard, W., and Roth, D., eds., *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-2011)*, 985–991.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-2011)*, 154–161.

Lu, Q.; Xu, Y.; Huang, R.; and Chen, Y. 2011. The Roamer planner random-walk assisted best-first search. In García-Olaya, A.; Jiménez, S.; and Linares López, C., eds., *The 2011 International Planning Competition*, 73–76.

Nakhost, H., and Müller, M. 2009. Monte-Carlo exploration for deterministic planning. In Walsh, T., ed., *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09)*, 1766–1771.

Nakhost, H., and Müller, M. 2012. A theoretical framework for studying random walk planning. In Borrajo, D.; Felner, A.; Korf, R. E.; Likhachev, M.; López, C. L.; Ruml, W.; and Sturtevant, N. R., eds., *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SOCS-2012)*, 57–64.

Nakhost, H.; Müller, M.; Valenzano, R.; and Xie, F. 2011. Arvand: the art of random walks. In García-Olaya, A.; Jiménez, S.; and Linares López, C., eds., *The 2011 International Planning Competition*, 15–16.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2012. Resource-constrained planning: A Monte Carlo random walk approach. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS-2012)*, 181–189.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (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.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In Fox, D., and Gomes, C. P., eds., *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, 975–982.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-2010)*, 246–249.

Valenzano, R.; Schaeffer, J.; Sturtevant, N.; and Xie, F. 2014. A comparison of knowledge-based GBFS enhancements and knowledge-free exploration. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS-2014)*.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-2004)*, 150–160.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-2014)*.

Xie, F.; Müller, M.; and Holte, R. 2014. Adding local exploration to greedy best-first search in satisficing planning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-2014)*.

Xie, F.; Nakhost, H.; and Müller, M. 2012. Planning via random walk-driven local search. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS-2012)*, 315–322.