Understanding and Improving Local Exploration for GBFS

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. The Greedy Best First Search with Local Search (GBFS-LS) algorithm adds exploration using a local GBFS to a global GBFS. This substantially improves performance for domains that contain large uninformative heuristic regions (UHR), such as plateaus or local minima.

This paper analyzes, quantifies and improves the performance of GBFS-LS. Planning problems with a mix of small and large UHRs are shown to be hard for GBFS but easy for GBFS-LS. In three standard IPC planning instances analyzed in detail, adding exploration using local GBFS gives more than three orders of magnitude speedup. As a second contribution, the detailed analysis leads to an improved GBFS-LS algorithm, which replaces larger-scale local GBFS explorations with a greater number of smaller explorations.

Introduction

Greedy Best First Search (GBFS) is the core search engine used in many state-of-the-art satisficing planners (Bonet and Geffner 2001; Helmert 2006; Lipovetzky and Geffner 2011; Xie, Müller, and Holte 2014b). GBFS always expands a node that minimizes a heuristic function h, without considering its g-value. GBFS can often find a solution quickly, but it might be of poor quality. When GBFS is used in a satisficing planner, an any-time policy is usually applied: the planner uses GBFS to find the first solution, and after that uses better quality search algorithms, such as Restarting Weighted-A* (RWA*) (Richter, Thayer, and Ruml 2010), to improve the plan quality.

The performance of GBFS strongly depends on the quality of the heuristic function. Domain-independent heuristic functions such as h^{FF} (Hoffmann and Nebel 2001) have known weaknesses that can lead to misleading heuristic estimates (Nakhost, Hoffmann, and Müller 2012; Xie et al. 2014; Imai and Kishimoto 2011).

One of the main problems of GBFS are Uninformative Heuristic Regions (UHR), which include plateaus and local minima. Xie, Müller, and Holte (2014a) propose a general framework, *GBFS with Local Exploration* (GBFS-LE), to attack this problem. Local exploration using random walks or local GBFS is invoked when global GBFS has failed to improve the heuristic value for a specified number of expansions.

# of node expansions needed for escaping (NEE)	number of nodes
[1,10]	456 (7.1%)
(10, 100]	66 (1.0%)
(100, 1000]	0 (0%)
>1000	5874 (91.8%)

Table 1: Number of nodes with different number of escaping node expansions.

One implementation of the GBFS-LE framework, GBFS with local GBFS (GBFS-LS), is the focus of the current paper. GBFS-LS differs from GBFS in the addition of a local GBFS, and was experimentally shown to yield a substantial improvement for IPC planning domains that have large UHRs. The current work analyzes the reasons for this improvement in detail.

The analysis will illustrate that a search method such as GBFS, which uses a global open list, can become stuck in the union of many distinct UHRs from different parts of the search space, which combine to form a large virtual UHR over the open list. This weakness of GBFS can be overcome by local exploration.

Example of a Large Virtual UHR

Instance #21 from the IPC-2004 pipesworld-notankage domain shows a clear case of such a large virtual UHR. Figure 1 plots the accumulated search time that GBFS and GBFS-LS need to reach the first node with a given minimum h^{FF}-value. GBFS requires almost 1000 seconds to decrease its minimum h-value from 2 to 1. GBFS-LS solves the whole problem in 2 seconds.

To understand the cause of this difference in performance, consider the following snapshot of GBFS: the minimum h-value (h_{min}) on the Open list is 2, there are 6396 nodes n on the Open list with $h(n) = 2$, and, since none of them has a child c with $h(c) < 2$, GBFS expands all 6396 nodes. In contrast, GBFS-LS expands only a small fraction of these nodes since some have a relatively quick local escape to a node n' with $h(n') < 2$. To quantify this, a small local GBFS was

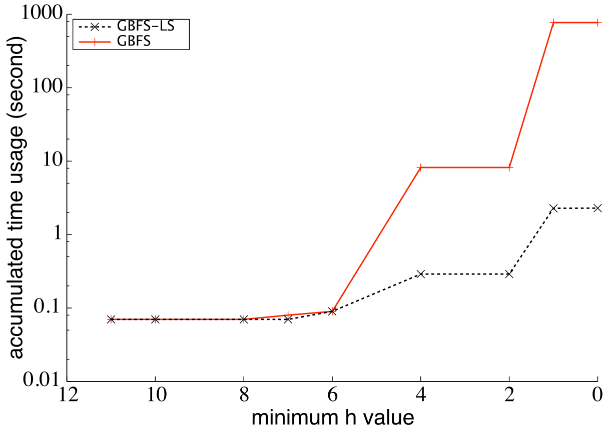


Figure 1: Search time (in seconds) of GBFS and GBFS-LS with h^{FF} for a given h-value (x-axis) to first become the minimum of all h-values generated up to that time. 2004-pipesworld-notankage instance #21.

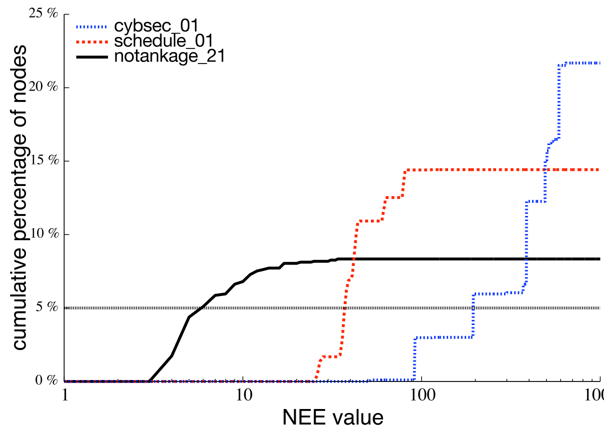


Figure 2: The distribution of NEE values over the 5000 picked nodes in 2000-Schedule #01, 2004-pipesworld-notankage#21, and 2008-cybersecurity #01.

started from each of these nodes n to determine NEE(n), the number of nodes expanded by a local GBFS search starting at n before a node with $h(n) < 2$ is encountered. Table 1 lists the order of magnitude of NEE(n) for these nodes. 7.1% of nodes, a non-negligible fraction, allow for a very quick escape, with $NEE(n) \leq 10$. In contrast, standard GBFS without local exploration requires 1.8 million node expansions to reach a node with $h(n) < 1$. This analysis illustrates the existence of small UHRs within a large virtual UHR and gives a hint of why local GBFS can substantially reduce the time required to reach a goal state.

Contributions and Organization of the Paper

The contributions of this work can be summarized as follows:

- Illustrate that there are both small UHRs and large UHRs in the open list of GBFS;
- Explain why adding local GBFS improves the performance;
- Show how to further improve the performance of GBFS based on the distribution of NEE values as in the example above.

The remainder of the paper is organized as follows. A more detailed analysis is presented to illustrate the existence of multiple UHRs and why GBFS-LS outperform the GBFS on three IPC instances. A modification of GBFS-LS, which further improves performance, is proposed and tested experimentally. The paper concludes with a discussion of possible future work.

The Problem of Simultaneous Expansion of Multiple Uninformative Heuristic Regions

This section investigates the problem of GBFS with multiple UHRs by analyzing search behaviour in three IPC planning instances: 2000-Schedule #10-0, 2004-pipesworld-notankage #21, and 2008-Cyber-security #01. h^{FF} is used as the heuristic. Experiments use one core of a 2.8 GHz Intel Xeon CPU machine with 4 GB memory and 30 minutes per instance.

In all three instances, GBFS gets stuck at $h_{min} = 2$. It fails to find a lower h-value in 30 minutes for 2000-Schedule #1, needs 1.8 million node expansions (1000 seconds) in 2004-pipesworld-notankage #21 to decrease h_{min} from 2 to 1, and needs 2.5 million node expansions in nearly 800 seconds to achieve the same step in 2008-Cybersecurity #1. GBFS-LS search time for completely solving these three instances is 28.26, 2.29 and 4.32 seconds respectively.

Small UHRs and Large UHRs

Given an expansion limit L (1000 in our experiments), a node n on the global open list is said to be a *small UHR* if a local GBFS search from n generates a node v with $h(v) < h_{min}$ after expanding L or nodes. i.e. $NEE(n) \leq L$. Otherwise, n is a *large UHR*.

The following experiments investigate the frequency of small and large UHRs in the open list of GBFS for the three planning instances above. The experiment begins when the first node n_1 with $h(n_1) = 2$ is added to the open list. The experiment has the following three steps:

1. Keep GBFS running for 10,000 *initializing expansions* in order to add more new nodes to the open list.
2. Define set S as the first 5000 nodes in the open list at this point in time. Use random tie-breaking to choose among nodes with equal h-value.
3. Run local GBFS from each node $n \in S$, setting $h = h^{FF}$, $h_{min} = 2$, $L = 1000$. Each local search uses an initially empty local open list and a local closed list initialized with the (fixed) global closed list. Both the local open list and the local closed list are discarded afterwards. The NEE of all nodes in S is recorded.

For the three test instances, Figure 2 shows the percentage of nodes with a NEE value less than or equal to the value on the x-axis. A non-negligible percentage of nodes with a relatively small NEE are found in all three instances. A local search starting from any of these nodes succeeds in reducing h_{min}. This result helps explain why GBFS-LS can quickly make progress, and can be orders of magnitude faster than GBFS.

The distribution of NEE varies among the three instances. For example, the value of x such that 5% of the nodes have a $NEE(n) \leq x$ is 6 for 2005-pipesworld-notankage #21, but is 38 for 2000-Schedule #01, and 195 for 2008-cybersecurity #01. For ease of comparison, a reference line with y=5% is shown in the figure. In each case, while the majority of the 5000 analyzed nodes seem to be located in large UHRs and cause the local search to fail, there is a sufficient number of nodes in small UHRs that local GBFS can make quick progress.

Besides the three domains for which examples were analyzed above, co-occurring small and large UHRs were detected in 10 further IPC domains: 1998-Mystery, 1998-Mystery-Prime, 2002-Depot, 2002-Driverlog, 2004-PSR-Large, 2006-Rovers, 2006-Storage, 2006-Pipesworld-Tankage, 2008-Scanalyzer and 2008-Transport. All these domains contain some instances where the GBFS expands a significantly larger number of nodes than the GBFS-LS in improving some h_{min} and meanwhile the GBFS-LS only expands a small number of nodes (less than 5,000) in finding such an improvement. In these instances, co-occurring small and large UHRs can be easily detected using the same method above.

Why does GBFS not explore Small UHRs?

In the cases analyzed above, enough small UHRs exist, from which local GBFS can easily escape. Why does global GBFS not find an escape path from these small UHRs? Is GBFS just unlucky, and always picks nodes with high NEE? The answer is no. Further analysis shows that all the escape paths found by local GBFS go through at least one *barrier* node n with $h(n) > h_{min}$. This means that all these small UHRs are local minima, not plateaus from which global GBFS could escape. While local GBFS can expand across such barrier nodes very quickly, GBFS is forced to exhaustively expand all nodes n' in all UHRs with $h(n') = h_{min}$, before it can expand any nodes with larger h.

As an extreme example, assume that all but one of the UHRs are local minima needing only 2 node expansions to escape from and that the other UHR contains a large number of nodes n' with $h(n') = h_{min}$. GBFS has to expand all the nodes in the large UHR before it can find an escape path from one of the others. The discussion here matches the observation by Xie, Müller, and Holte (2014a) that GBFS-LS only improves the performance in domains that contain large UHRs. Such barrier nodes also exist for other two commonly used heuristics: causal graph (CG) (Helmert 2004) and context-enhanced additive (CEA) (Helmert and Geffner 2008).

However, not all small UHRs must be local minima. As an example, in 2006-Pipesworld-Tankage instance #32, GBFS needs 4706 node expansions in 13 seconds to improve h_{min} from 6 to 5, while it takes GBFS-LS 6.6 seconds and 1621 node expansions starting from the same open list. The virtual UHR over the open list contains both local minima and plateaus. Because the global GBFS can escape from the plateaus, GBFS-LS does not improve the performance as dramatically as it does in the three instances above.

More Exploration with Smaller Local GBFS

Greedy Best First Search with Local Search (GBFS-LS) (Xie, Müller, and Holte 2014a) is the same as GBFS except it executes a local GBFS whenever the global GBFS (G-GBFS) seems stalled. G-GBFS is considered *stalled* if it fails to improve its minimum heuristic value h_{min} for a specified number *STALL_SIZE* of node expansions, set to 1000 by default. In this case GBFS-LS runs a small local GBFS for exploration, from a best node *n* in the (global) open list. After each local exploration, the mechanism for detecting stalled global search is reset.

Local GBFS can dramatically improve the time to solution, if used for nodes in small UHRs. In the original GBFS-LS algorithm from Xie, Müller, and Holte's work (2014a), a single local GBFS is called and expands up to 1000 nodes whenever G-GBFS did not improve h_{min} over its last 1000 expansions. Therefore each single local GBFS is relatively expensive.

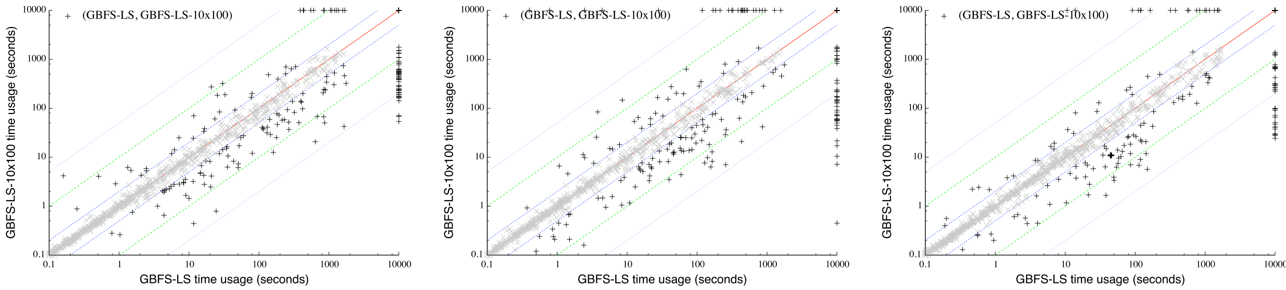
The experiments above suggest that using more frequent but smaller local searches may be a good tradeoff. To investigate this with minimal changes to the algorithm, the proposed new scheme GBFS-LS-$X \times Y$, where $X \times Y = 1000$, runs X local searches with Y expansions each from X random nodes in the best (minimum h) bucket in the open list. If there are fewer than X nodes in this bucket, the remaining nodes are chosen from the next-best bucket(s). Our experiments include the following pairs of (X, Y): (1, 1000), (10, 100), (100, 10) and (1000, 1).

Experiments were run on the same set of 2112 problems as in Xie, Müller, and Holte's work (2014a), in 54 domains from the first seven International Planning Competitions (IPC 1 to 7), using one core of a 2.8 GHz Intel Xeon CPU 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 Jasper (Xie, Müller, and Holte 2014b) code base, which is downloaded from the IPC-8 website.[1] The translation from PDDL to SAS$^+$ was done only once, and this common preprocessing time is not counted in the 30 minutes.

Table 2 compares the new algorithms with GBFS and GBFS-LS. 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 2 shows the coverage on all 2112 IPC instances. Overall, GBFS-LS-10$\times$100 outperforms GBFS, GBFS-LS and other configurations for all three heuristics.

GBFS-LS-1$\times$1000 and GBFS-LS-1000$\times$1 are added for evaluating the influence of randomness. While GBFS-LS applies a deterministic first-in-first-out approach in picking the starting node for local GBFS, GBFS-LS-1$\times$1000 applies the random tie-breaking. However, these two algorithms achieve very similar coverage results. Similarly, GBFS-LS-1000$\times$1

[1] http://helios.hud.ac.uk/scommv/IPC-14/



(a) FF heuristic (b) CG heuristic (c) CEA heuristic

Figure 3: Comparison of search time of GBFS-LS-10×100 with GBFS-LS for three different heuristics. For each heuristic, we compared all 5 runs results of GBFS-LS-10×100 with GBFS-LS, and one typical run is selected.

Heuristic	GBFS	GBFS-LS	GBFS-LS-1×1000	GBFS-LS-10×100	GBFS-LS-100×10	GBFS-LS-1000×1
FF	1561	1641	1641.0	**1678.2**	1659.2	1576.4
CG	1513	1608	1600.2	**1618.4**	1595.2	1516.4
CEA	1498	1592	1577.2	**1612.4**	1609.8	1497.0

Table 2: IPC coverage out of 2112 for GBFS, GBFS-LS, GBFS-LS-1×1000, GBFS-LS-10×100, GBFS-LS-100×10 and GBFS-LS-1000×1. three standard heuristics.

is very close to a GBFS version that applies the random tie-breaking, which also results in a very similar coverage result to GBFS. These two data points show that the superior performance of GBFS-LS-10×100 over GBFS-LS is due to running a larger number of small local searches and not due to the randomness in the node selection process.

Figure 3 compares the search time of GBFS-LS-10×100 with GBFS-LS over the IPC benchmarks. Every point in the figure represents one problem instance, with the search time for GBFS-LS on the instance plotted on the x-axis and the time for GBFS-LS-10×100 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 GBFS-LS-10×100 solves faster than GBFS-LS. 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. For all the heuristics tested, besides its improved coverage, GBFS-LS-10×100 also shows a substantial improvement in search time over GBFS-LS, with many more results in the factor 2 to 10 speed up region favouring the new algorithm.

The same modification was also tested with LAMA-LS, which replaces the GBFS component of LAMA-2011 with GBFS-LS (Xie, Müller, and Holte 2014a). Unfortunately, there is no noticeable improvement here: LAMA-LS and LAMA-LS-10×100 are very similar in both coverage and search time. One possible reason is that the major enhancements in LAMA-2011 such as deferred evaluation, preferred operators (Richter and Helmert 2009) and multiple heuristics (Richter and Westphal 2010), already cover some bad scenarios for GBFS-LS. This is a topic for future study.

Conclusion and Future Work

This paper illustrates the multiple UHRs problem of GBFS using three IPC examples, and explains why adding local GBFS improves the performance of GBFS. As suggested by the analysis, it is confirmed that running a larger number of smaller local searches further improves the performance.

Future work should explore whether the same problem occurs in classical heuristic search domains, such as sliding tile puzzles (Korf and Taylor 1996). Valenzano et al. (2014) show that replacing preferred operators with random actions can achieve about half the improvement of preferred operators. Similarly, replacing the secondary heuristic in multiple heuristics with a purely random heuristic achieves about half the improvement of multiple heuristics. The multiple UHR problem might be a contributing cause of these two phenomena.

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.

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.

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.

Korf, R. E., and Taylor, L. 1996. Finding optimal solutions to the twenty-four puzzle. In Clancey, W. J., and Weld, D. S., eds., *The Thirteenth National Conference on Artificial Intelligence*, 1202–1207.

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.

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.; Thayer, J.; and Ruml, W. 2010. The joy of forgetting: Faster anytime search via restarting. 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)*, 137–144.

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 Twenty-Fourth International Conference on Automated Planning and Scheduling*, 375–379.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2395–2402.

Xie, F.; Müller, M.; and Holte, R. 2014a. Adding local exploration to Greedy Best-First Search in satisficing planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2388–2394.

Xie, F.; Müller, M.; and Holte, R. 2014b. Jasper: the art of exploration in Greedy Best First Search. In Vallati, M.; Chrpa, L.; and McCluskey, T., eds., *The Eighth International Planning Competition (IPC-2014)*, 39–42.