Jasper: the Art of Exploration in Greedy Best First Search

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

Introduction

LAMA-2011 (Richter and Westphal 2010) is the clear winner of the sequential satisficing track in the latest International Planning Competition (IPC-2011). It finds a first solution by Greedy Best-First Search (GBFS), and then continues to improve solutions using restarting weighted A* (Richter, Thayer, and Ruml 2010). *Diverse Anytime Search (DAS)* (Xie, Valenzano, and Müller 2013) is a meta-algorithm designed for solution improvement. It takes an anytime planner and a post-processing system, and adds restarts and randomization for better quality search.

Jasper is a satisficing planner that builds on LAMA-2011. It adds two modifications. First, it replaces the GBFS algorithm in LAMA-2011 with an improved GBFS variant, called *Type Exploration based Greedy Best-First Search with Local Search (Type-GBFS-LS)*. GBFS always expands a node n that is closest to a goal state according to a heuristic h. GBFS' performance strongly depends on h. Uninformative or misleading heuristics can massively increase the time and memory complexity of such searches. Type-GBFS-LS is an improved version of GBFS that is less sensitive to such flaws in heuristic functions. Second, it implements the DAS system for solution improvement, which takes the modified LAMA-2011 as the anytime planner and Aras (Nakhost and Müller 2010) as the post-processing system.

A detailed description of the implementation of DAS can be found in the ICAPS paper by Xie, Valenzano and Müller (Xie, Valenzano, and Müller 2013). This paper focuses on describing the new search algorithm, Type-GBFS-LS.

The remainder of this paper is organized as follows. First, we motivate this work by discussing the two potential problems of GBFS: *uninformative heuristic region and misleading heuristics*, followed by describing two corresponding solutions as well as their combination, Type-GBFS-LS. Later, experimental results show that the proposed algorithms improve the state of the art planner LAMA-2011 significantly.

Uninformative Heuristic Regions (UHR) and GBFS with Local Search

The notion of an Uninformative Heuristic Region (UHR) includes both *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.

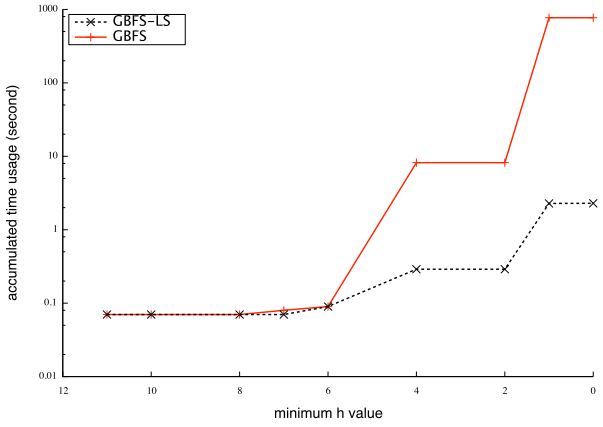


Figure 1: Cumulative search time (in seconds) of GBFS, and GBFS-LS with h^{FF} for first reaching a given h_{min} in 2004-notankage #21.

As an example, the IPC domain 2004-notankage has no dead ends, but contains large plateaus and local minima (Hoffmann 2011). Instance #21 shown in Figure 1 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 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 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.

GBFS with Local Search

The new algorithm of *Greedy Best-First Search with Local Search (GBFS-LS)* uses a local GBFS (*LS*) 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-LS runs a small local GBFS for exploration from the best node n in a global-level open list.

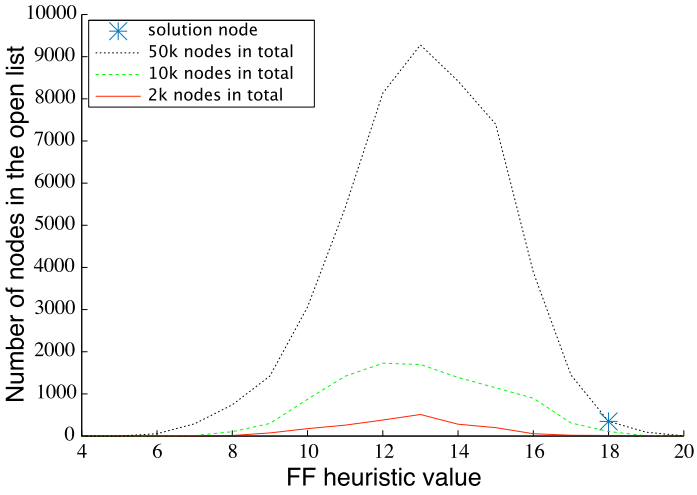
LS shares the closed list of G-GBFS, but maintains its own separate open list *local_open* that is cleared before each local search. LS succeeds if it finds a new best node v with $h(v) < h_{min}$ before it exceeds a given limit on the number of nodes. In any case, the remaining nodes in *local_open* 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* use a first-in-first-out tie-breaking rule. A detailed description of GBFS-LS can be found in (Xie, Müller, and Holte 2014). In Figure 1, the same problem takes GBFS-LS only 1 second to solve, while it takes the basic GBFS around 1000 seconds.

Misleading Heuristics (ML) and Type Exploration in GBFS

Early mistakes are mistakes in search direction at shallow levels of the search tree, caused by sibling nodes being expanded in the wrong order due to a misleading heuristic. the root node of a *bad subtree*, which contains no solution or only hard-to-find solutions, has a lower heuristic value than a sibling which would lead to a quick solution.

The 2011-Nomystery domain from IPC-2011 is a typical example where delete-relaxation heuristics systematically make early mistakes (Nakhost, Hoffmann, and Müller 2012). In this transportation domain with limited non-replenishable fuel, delete-relaxation heuristics such as h^{FF} ignore the crucial aspect of fuel consumption, which makes the heuristic overoptimistic and misleading. Bad subtrees in the search tree, which over-consume fuel early on, are searched exhaustively, before any good subtrees which consume less fuel and can lead to a solution are explored. As a result, while the random walk-based planner Arvand with its focus on exploration solved 19 out of 20 nomystery instances in IPC-2011, LAMA-2011 solved only 10.

Previous exploration methods in GBFS suffer from biasing their exploration heavily towards the neighborhood of nodes in the open list. In the case of early mistakes, the large majority of these nodes is in useless regions of the search space. Consider the nodes in the regular h^{FF} open list of LAMA-2011 while solving the problem 2011-nomystery #12. Figure 2 shows snapshots of their h-value distribution after 2,000, 10,000 and 50,000 nodes expanded. In the figure, the x-axis represents different heuristic values and the y-axis represents the number of nodes with a specific h value in the open list. The solution eventually found by LAMA-2011 goes through a single node n in this 50,000 node list, with $h(n) = 18$. This node is marked with a star in the figure. Over 99% of the nodes in the open list have lower h-values, and will be expanded first, along with much of their subtrees. However, in this example, none of those nodes leads to a solution. The open list is flooded with a large number of very similar, useless nodes from undetected dead ends



h-values in 2011-nomystery #12

Figure 2: h-value distribution in the regular h^{FF} open list of LAMA-2011.

or local minima.

ϵ-GBFS (Valenzano et al. 2014) samples nodes uniformly over the whole open list. This is not too useful when entries are heavily clustered in bad subtrees. In the example above, ϵ-GBFS has a less than 1% probability to pick a node with h-value 18 or more in its exploration step, which itself is only executed with probability ϵ. Furthermore, the algorithm must potentially select several good successor nodes before making measurable progress towards a solution by finding an exit node with a lower h-value.

Type System

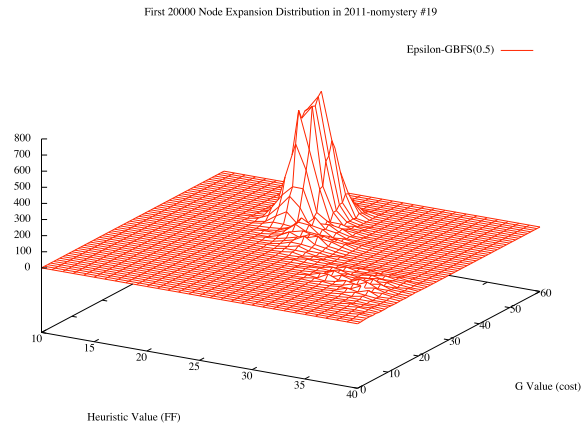
Can the open list be sampled in a way that avoids the over-concentration on a cluster of very similar nodes? A *type system* (Lelis, Zilles, and Holte 2013), which is based on earlier ideas of stratified sampling (Chen 1992), is one possible approach. It is defined as follows:

Definition 1 *(Lelis, Zilles, and Holte 2013) Let S be the set of nodes in search space. $T = \{t_1, \ldots, t_n\}$ is a type system for S if T is a disjoint partitioning of S. For every $s \in S$, $T(s)$ denotes the unique $t \in T$ with $s \in S$.*

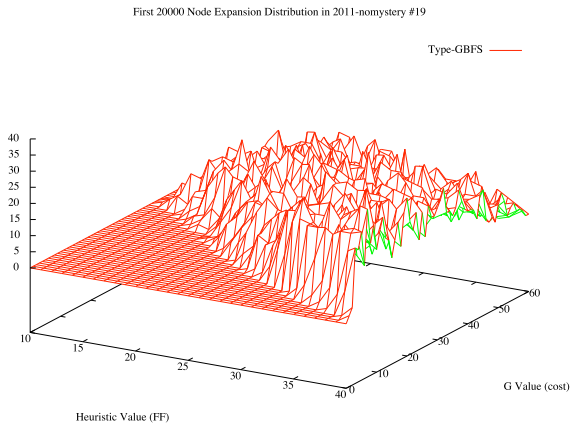
Types can be defined using any property of nodes. The simple type system used here defines the type of a node s in terms of its h-value for different heuristics h, and its g-value. A simple and successful choice is the pair $T(s) = (h_{FF}(s), g(s))$. The intuition behind such type systems is that they can roughly differentiate between nodes in different search regions, and help explore regions differents from the nodes where GBFS gets stuck.

Type-GBFS: Adding a Type System to GBFS

Type-GBFS uses a simple two level *type bucket* data structure tb which organizes its nodes in buckets according to their type. Type bucket-based node selection works as follows: first, pick a bucket b uniformly at random from among



(a) ϵ-GBFS($\epsilon = 0.5$)



(b) Type-GBFS

Figure 3: Distribution of types over the first 20,000 nodes expanded by the exploring phase (ϵ-exploration or type buckets) of ϵ-GBFS($\epsilon = 0.5$) and Type-GBFS.

all non-empty buckets and then pick a node n uniformly at random from all nodes in b. Type-GBFS alternately expands a node from the regular open list O and from tb, and each new node is added to both O and tb. A detailed description of Type-GBFS can be found in (Xie et al. 2014).

Type-GBFS and ϵ-GBFS with $\epsilon = 0.5$ both spend half their search effort on exploration. However, the distribution of types of the explored nodes is very different. Figure 3 shows the frequency of explored node types for ϵ-GBFS with $\epsilon = 0.5$ and Type-GBFS[1] after 20,000 nodes in the same format. ϵ-GBFS mainly explores nodes close to the low heuristic value types, while Type-GBFS explores much more uniformly over the space of types.

Note that the z-axis scales are different for the two figures. The single most explored type contains around 800 nodes for ϵ-GBFS but only 40 for Type-GBFS. The presence or absence of exploration helps explain the relative performance in 2011-Nomystery. The coverage for the 20 instances of this domain for one typical run under IPC conditions is 9 for GBFS, 11 for ϵ-GBFS with $\epsilon = 0.5$, and 17 for Type-GBFS.

Combining GBFS-LS and Type-GBFS

GBFS-LS and Type-GBFS are designed for two different problems in GBFS. Jasper applies both enhancements to GBFS. The new algorithm is called *Type Exploration based Greedy Best-First Search with Local Search (Type-GBFS-LS)*. Like GBFS-LS, Type-GBFS-LS uses a local search when the global search gets stuck. However, it replaces GBFS with Type-GBFS in both the global level search and the local level search.

Experiments

Experiments were run on a set of 2112 problems in 54 domains from the seven International Planning Competitions, 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.

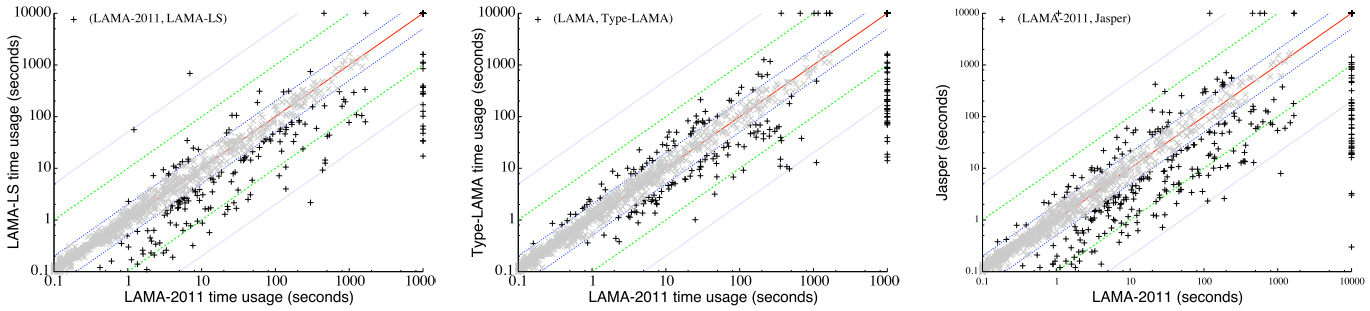
The performance comparison in this section includes the following planners:

- **LAMA-2011:** only the first iteration of LAMA using GBFS is run, with deferred evaluation, preferred operators and multi-heuristics (h^{FF}, h^{lm}) (Richter and Westphal 2010).
- **LAMA-LS:** Configured like LAMA-2011, but with GBFS replaced by GBFS-LS.
- **Type-LAMA:** With GBFS replaced by Type-GBFS, uses the same four queues as LAMA-2011, plus (h^{FF}, g) type buckets.
- **Jasper:** Configured like LAMA-2011, but with GBFS replaced by Type-GBFS-LS. It uses the same four queues as LAMA-2011 plus (h^{FF}, g) type buckets in both the global search and the local search.

Table 1 shows the coverage results for the four planners. All the three proposed planners get better results than LAMA-2011, with the best result of 1953.0 for Jasper.

Each diagram in Figure 4 compares one planner with LAMA-2011 on their time performance. Every data point represents one instance, with the search time for LAMA-2011 on the x-axis plotted against the corresponding planner 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 Type-GBFS solves 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 greyed out 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 $x = 10000$ and $y = 10000$.

[1] Some explored types are outside the (h, g) range shown in Figure 3 (b).



(a) LAMA-2011 (x) vs. LAMA-LS(y) (b) LAMA-2011 (x) vs. Type-LAMA (y) (c) LAMA-2011 (x) vs. Jasper (y)

Figure 4: Comparison of search time: LAMA-2011 vs. LAMA-LS (a), Type-LAMA(b) and Jasper.

All the three proposed planners show a clear overall improvement over LAMA-2011 in terms of speed. Jasper has the best overall performance. It solves more problems than LAMA-LS. Besides its advantage in coverage, it wins the time comparison with Type-LAMA for a larger number of instances by factors 2x and 10x.

Planner	LAMA-2011	LAMA-LS	Type-LAMA	Jasper
Coverage	1913	1931	1949.8	**1953.0**

Table 1: IPC coverage out of 2112.

References

Benton, J.; Haslum, P.; Helmert, M.; Katz, M.; and Thayer, J., eds. 2014. *Proceedings of the Sixth Workshop on Heuristic Search for Domain-Independent Planning, HSDIP 2014*.

Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds. 2010. *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*.

Chen, P. C. 1992. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM J. Comput.* 21(2):295–315.

Hoffmann, J. 2011. Where ignoring delete lists works, part II: Causal graphs. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *ICAPS*, 98–105. AAAI.

Lelis, L. H. S.; Zilles, S.; and Holte, R. C. 2013. Stratified tree search: a novel suboptimal heuristic search algorithm. In Gini, M. L.; Shehory, O.; Ito, T.; and Jonker, C. M., eds., *AAMAS*, 555–562.

Nakhost, H., and Müller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In Brafman et al. (2010), 121–128.

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., *Proceeedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS-2012)*, 181–189.

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 et al. (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 *ICAPS*.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In Benton et al. (2014). 9 pages.

Xie, F.; Müller, M.; and Holte, R. 2014. Adding local exploration to greedy best-first search in satisficing planning. In Benton et al. (2014). 9 pages.

Xie, F.; Valenzano, R.; and Müller, M. 2013. Better time constrained search via randomization and postprocessing. In *ICAPS*, 269–277. AAAI.