Type-Based Exploration with Multiple Search Queues for Satisficing Planning

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

Tatsuya Imai
Tokyo Institute of Technology
Tokyo, Japan
imai7@is.titech.ac.jp

Abstract

Utilizing multiple queues in Greedy Best-First Search (GBFS) has been proven to be a very effective approach to satisficing planning. Successful techniques include extra queues based on *Helpful Actions* (or *Preferred Operators*), as well as using *Multiple Heuristics*. One weakness of all standard GBFS algorithms is their lack of exploration. All queues used in these methods work as priority queues sorted by heuristic values. Therefore, misleading heuristics, especially early in the search process, can cause the search to become ineffective.

Type systems, as introduced for heuristic search by Lelis et al, are a development of ideas for exploration related to the classic stratified sampling approach. The current work introduces a search algorithm that utilizes type systems in a new way – for exploration within a GBFS multiqueue framework in satisficing planning.

A careful case study shows the benefits of such exploration for overcoming deficiencies of the heuristic. The proposed new baseline algorithm *Type-GBFS* solves almost 200 more problems than baseline GBFS over all International Planning Competition problems. Type-LAMA, a new planner which integrates Type-GBFS into LAMA-2011, solves 36.8 more problems than LAMA-2011.

Introduction

In the latest International Planning Competition (IPC) 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 (Richter and Helmert 2009), Deferred Evaluation (Richter and Helmert 2009) and Multi-Heuristic (Richter and Westphal 2010).

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. GBFS's performance strongly depends on h. Misleading or uninformative heuristics can result in massive increases in the time and memory complexity of search.

Two of the three enhancements above, Preferred Operators and Multi-Heuristic, are implemented in a *Multiple Queue Search* framework (Helmert 2006). Separate priority queues are used to hold different sets of nodes, or keep them sorted according to different heuristics. Still, each queue is sorted based on some heuristic h, and is used in a greedy fashion by the search, which always expands a node with minimum h-value from one of the queues. This makes search vulnerable to the *misleading heuristic* problem, where it can stall in *bad subtrees*, which contain large local minima or plateaus but do not lead to a solution. Adding exploration to a search algorithm is one way to attack this problem.

Previous approaches to this problem of GBFS with misleading heuristics include K-BFS (Felner, Kraus, and Korf 2003), which expands the first k best nodes in a single priority queue and adds all their successors, and Diverse-BFS (Imai and Kishimoto 2011), which expands extra nodes with non-minimal h-values or at shallow levels of the search tree. Another simple algorithm is ϵ-GBFS (Valenzano et al. 2014), which expands a node selected uniformly at random from the open list with probability ϵ. All these algorithms add an element of exploration.

The current paper proposes and evaluates a simple yet very effective way of adding exploration based on a *type system* (Lelis, Zilles, and Holte 2013). The major contributions are:

1. An analysis of the weaknesses of previous simple exploration schemes. and a non-greedy approach to exploration based on a simple type system.
2. A search algorithm under the framework of multiple-queue search named *Type-GBFS* which uses a type system for exploration, and the corresponding planner *Type-LAMA*, which replaces the GBFS component of LAMA-2011 by Type-GBFS.
3. Detailed experiments on IPC benchmarks, which demonstrate that baseline Type-GBFS solves substantially more problems than baseline GBFS, and that this superiority also holds when adding most combinations of standard planning enhancements. Type-LAMA with all such enhancements outperforms LAMA-2011.

The discussion starts with a review of Multiple Queue Search and the problems caused by misleading heuristics in

best-first algorithms. An analysis of ϵ-GBFS shows that its exploration suffers from closely following the distribution of h-values in the open list. A simple type system based on both g- and h-values of each node is proposed as a remedy and shown to lead to better exploration. Experiments on the baseline as well as state of the art planners confirm that this leads to significant progress in coverage.

Background: Multiple Queue Search and Early Mistakes in GBFS

Multiple Queue Search

Two of the most important enhancements for satisficing planning, Preferred Operators and Multi-Heuristics, are based on *Multiple Queue Search* (Helmert 2006; Röger and Helmert 2010). When more than one heuristic is used in the search algorithm, Multiple Queue Search uses one priority queue for each heuristic, and selects the next node to expand from these queues in a round-robin manner. Queue boosting (Helmert 2006) can be used to assign priorities to different queues. Once a node is expanded, all its successors are evaluated by all heuristics, and put into every queue with the corresponding value. For Preferred Operators (or Helpful Actions), one additional priority queue per heuristic is used, which contains only the successors generated by a preferred operator. The algorithm again selects nodes in a round-robin manner from all queues, but boosts the preferred operator queue(s). The search benefits both from focusing more on usually relevant actions, and from reaching greater depths more quickly because of the smaller effective branching factor in the preferred queues.

Early Mistakes caused by Misleading Heuristics

Early mistakes are mistakes in search direction at shallow levels of the search tree caused by sibling nodes being expanded in the wrong order. This happens when 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, and results in large unrecognized dead-ends in the search space. 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.

Exploration bias in the Open List: Two Case Studies

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} (Hoffmann and Nebel 2001) open list of LAMA-2011 while solving the problem 2011-nomystery #12. Figure 1(a) 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 an asterisk 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 useless nodes with undetected dead ends.

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.

The instance 2011-nomystery #12, with 6 locations and 6 packages, has a relatively small search space, and both GBFS and ϵ-GBFS eventually solve it after exhaustively enumerating the dead ends. However, a larger problem like 2011-nomystery #19, with 13 locations and 13 packages, is completely out of reach for GBFS or ϵ-GBFS. This instance was solved by only 2 planners in IPC-2011. Figure 1(b) shows the h-value distribution in LAMA-2011's regular h^{FF} queue after 20,000, 100,000 and 500,000 nodes. The node with $h = 39$ from a solution found by Arvand-2011 (Nakhost and Müller 2009) is marked at the far right tail of the distribution in the figure.

Adding Exploration via a Type System

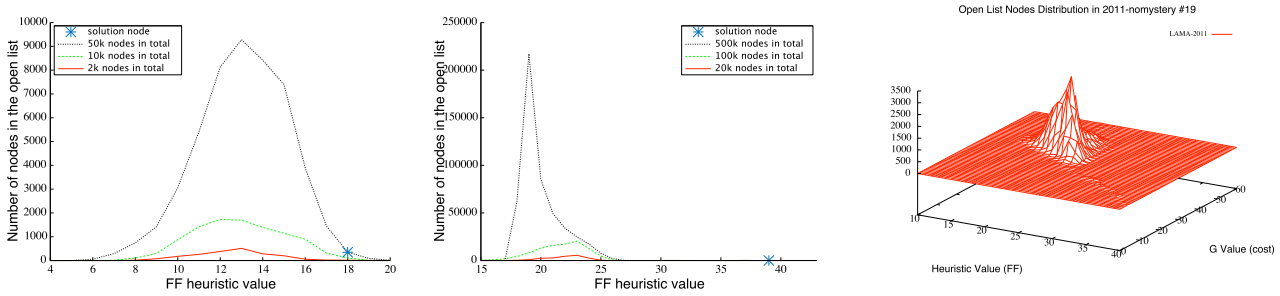
Can the open list be sampled in a way that avoids the over-concentration on a cluster of very similar nodes? A *type system* (Lelis, Zilles, and Holte 2013), which is based on earlier ideas of stratified sampling (Chen 1992), is one possible approach.

Type System

A *type system* 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



(a) h-values in nomystery #12 (b) h-values in nomystery #19 (c) (h, g)-distribution in nomystery #19

Figure 1: (a)(b): h-value distribution in the regular h^{FF} open list of LAMA-2011. (c): distribution of types in the regular h^{FF} open list of LAMA-2011 after 100,000 nodes in 2011-nomystery #19.

that they can roughly differentiate between nodes in different search regions, and help explore away from the nodes where GBFS gets stuck.

Figure 1(c) views a LAMA-2011 search of instance 2011-nomystery #19 through the lens of a (h^{FF}, g) type system. The horizontal x- and y-axes represent h^{FF}-values and g-values respectively. The number of nodes in the open list with a specific (h^{FF}, g) type is plotted on the vertical z-axis. The graph shows the frequency of each type in the regular h^{FF} open list of LAMA-2011 at the time when the open list first reaches 100,000 nodes. After initial rapid progress, search has stalled around a single huge peak. Most of the open list is filled with a large number of useless nodes.

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 all the non-empty buckets. Then pick a node n uniformly at random from all the nodes in b. Type-GBFS alternately expands a node from from the regular open list O and from tb, and each new node is added to both O and tb.

Multi-Heuristic type systems $(h_1(s), h_2(s), ...)$ have been explored before using a *Pareto set* approach (Röger and Helmert 2010). The main differences of their approach are: 1) only *Pareto Optimal* buckets are selected; 2) the probability of selecting each Pareto optimal bucket is proportional to the number of nodes it contains; 3) only heuristics are used to define types, whereas the current approach also considers g and potentially any other relevant information; and 4) nodes in a bucket are selected deterministically in FIFO order, not uniformly at random.

Diverse Best-First Search (DBFS) (Imai and Kishimoto 2011) is another closely related high performance search algorithm which includes an exploration component. This two-level search algorithm uses a global open list O_G, a local open list O_L and a shared closed list.

Like Type-GBFS with the $(h_{FF}(s), g(s))$ type system, DBFS picks nodes based on their h- and g-values. There are three major differences between these algorithms. 1) DBFS performs a sequence of local searches while Type-GBFS defines a single global search; 2) DBFS uses g to restrict its node selection, while Type-GBFS can use g as part of its type system; 3) DBFS biases its node selection using h, while Type-GBFS samples uniformly over all types.

Exploration in Type-GBFS, ϵ-GBFS and DBFS

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. In Nomystery-2011 #19, GBFS in LAMA-2011 grows the single peak shown in Figure 1(c). Figure 2 shows the frequency of explored node types for ϵ-GBFS with $\epsilon = 0.5$, Type-GBFS[1] and DBFS after 20,000 nodes in the same format. Note that while Figure 1 shows the distribution of all nodes in the open list, Figure 2 shows the types of only those nodes that were chosen in the exploration step[2].

ϵ-GBFS mainly explores nodes close to the GBFS peak types, while Type-GBFS explores much more uniformly over the space of types. DBFS explores more types than ϵ-GBFS. Unlike Type-GBFS, which samples types uniformly, DBFS is still biased towards low h and high g values.

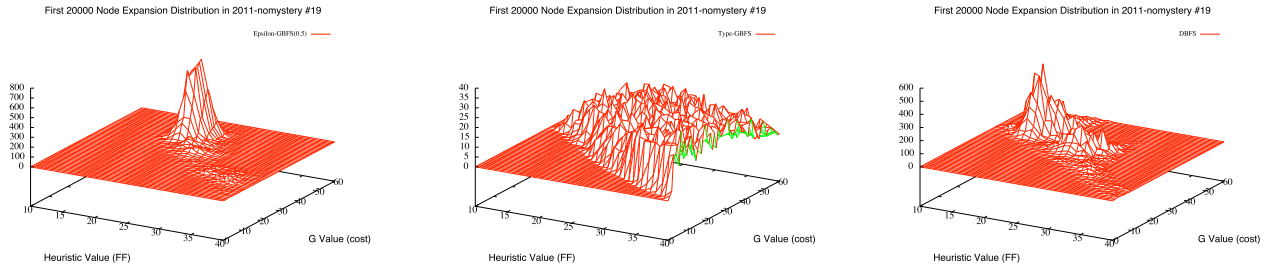
Note that the z-axis scales are different for the three plots. The single most explored type contains around 800 nodes for ϵ-GBFS and 600 for DBFS, but only 40 for Type-GBFS. The presence or absence of exploration helps explain the relative performance in 2011-Nomystery. The coverage for the 20 instances of this domain for one typical run with 4 GB memory and 30 minutes per instance is 9 for GBFS, 11 for ϵ-GBFS with $\epsilon = 0.5$, 17 for Type-GBFS and 18 for DBFS. While ϵ-GBFS slightly improves over GBFS, Type-GBFS outperforms both other algorithms. DBFS's exploration strategy also performs very well in 2011-Nomystery.

Experiments

The experiments use a set of 2112 problems (54 domains) from the seven International Planning Competitions, and were run on an 8-core 2.8 GHz machine with 4 GB memory

[1] Some explored types are outside the (h, g) range shown in Figure 2(b).

[2] Unlike ϵ-GBFS and Type-GBFS, there is no clear exploration step in DBFS. All visited nodes are shown in the figure.



(a) ϵ-GBFS($\epsilon = 0.5$) (b) Type-GBFS (c) DBFS

Figure 2: Distribution of types over the first 20,000 nodes expanded in the exploring phase (ϵ-exploration or type buckets) of ϵ-GBFS($\epsilon = 0.5$), Type-GBFS and DBFS.

and 30 minutes per instance. Results for planners which use randomization are averaged over five runs. All algorithms are implemented using 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.

Performance of Baseline Algorithms

The baseline study evaluates the two algorithms GBFS and Type-GBFS without the common planning enhancements of preferred operators, deferred evaluation and multi-heuristics. It uses three popular planning heuristics - FF (Hoffmann and Nebel 2001), causal graph (CG) (Helmert 2004) and context-enhanced additive (CEA) (Helmert and Geffner 2008). Table 1 shows the coverage results. Type-GBFS outperforms GBFS by a substantial margin for each tested heuristic.

Heuristic	GBFS	Type-GBFS
FF	1561	**1755.6**
CG	1513	**1691.4**
CEA	1498	**1678.8**

Table 1: Baseline GBFS vs. Type-GBFS - coverage of 2112 IPC instances.

Figure 3(a) compares the time performance of the baseline algorithms with h^{FF}. Every data point represents one instance, with the search times for GBFS on the x-axis plotted against Type-GBFS 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 $x = 10000$ or $y = 10000$.

Beyond solving almost 200 more problems, Type-GBFS shows a substantial advantage over GBFS in search time for problems solved by both planners. There are many more problems where Type-GBFS outperforms GBFS by more than a factor of 10 or 50 than vice versa. Still, while Type-GBFS outperforms GBFS overall, it does not dominate it on a per-instance basis. Sometimes the extra exploration of Type-GBFS wastes time or even leads the search astray for a while.

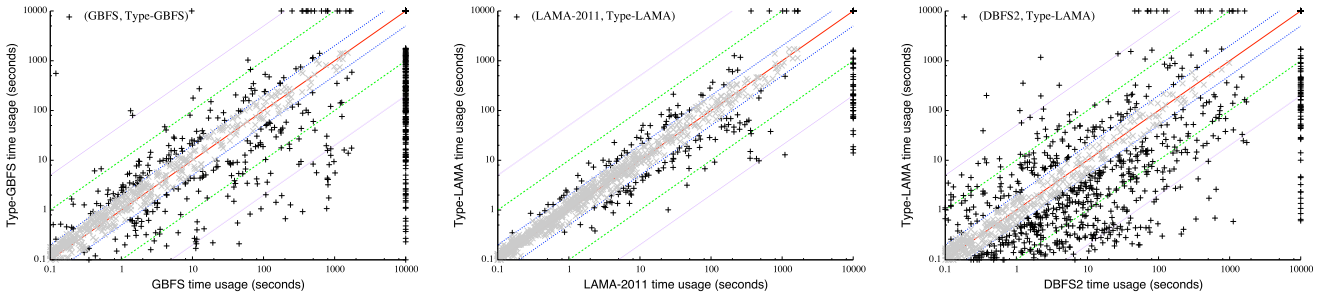
Performance with Different Enhancements

How do GBFS and Type-GBFS compare when common planning enhancements are added? All combinations of Deferred Evaluation, Preferred Operators and Multiple Heuristics are tested, with h^{FF} as the primary heuristic.

- With **Deferred Evaluation**, nodes are not evaluated before adding them to open lists and type buckets. Instead, the heuristic value of their parents is used (Richter and Helmert 2009).
- With **Preferred Operators**, nodes that are reached via preferred operators, such as helpful actions in h^{FF}, are also stored in a separate open list (Richter and Helmert 2009). Boosting of preferred open lists with a parameter of 1000 is used as in LAMA-2011 (Richter and Westphal 2010). In case of deferred evaluation, preferred operators are ranked before other siblings for tie-breaking, using the so-called *pref_first* ordering (Richter and Westphal 2010). Type-GBFS uses only a single set of type buckets for all nodes. There are no separate type buckets containing preferred nodes only.
- **Multi-Heuristics** maintains multiple priority queues sorted by different heuristics. Following LAMA, the *Landmark count* heuristic h^{lm} (Richter, Helmert, and Westphal 2008) is used as the second heuristic here. Type-GBFS with Multi-Heuristics uses two open lists, one for each heuristic, plus type buckets for the (h^{FF}, g) type system.

When both Multi-Heuristic and Preferred Operators are applied, GBFS uses four queues, two regular and two preferred ones. Type-GBFS uses the same queues plus (h^{FF}, g) type buckets.

Table 2 shows the experimental results on IPC domains for all 8 combinations of enhancements. When used as a single enhancement, Preferred Operators and Multiple Heuristic improve both algorithms. Deferred Evaluation also



(a) GBFS (x) vs. Type-GBFS(y) (b) Lama-2011 (x) vs. Type-LAMA (y) (c) DBFS2 (x) vs. Type-LAMA (y)

Figure 3: Comparison of search time. (a): GBFS and Type-GBFS with the h^{FF} heuristic on IPC. (b)(c): Type-LAMA vs. Lama-2011 (left) and DBFS2 (right). using typical single runs of Type-GBFS, Type-LAMA and DBFS2.

Enhancement	GBFS	Type-GBFS
(none)	1561	1755.6
PO	1826	1848.6
DE	1535	1834.6
MH	1851	1789.8
PO + DE	1871	1906.4
PO + MH	1850	1846.2
DE + MH	1660	1729.0
PO + DE + MH	**1913**	**1949.8**

Table 2: Number of IPC tasks solved out of 2112. PO = Preferred Operators, DE = Deferred Evaluation, MH = Multi-Heuristic.

strongly improves Type-GBFS, but causes a slight decrease in coverage for GBFS, mainly due to plateaus caused by the less informative node evaluation (Richter and Helmert 2009). Apparently, Type-GBFS gets stuck in such plateaus less often.

When combining any two enhancements, both algorithms achieve their best performance with Preferred Operators plus Deferred Evaluation, as observed for GBFS in (Richter and Helmert 2009). Multiple Heuristics have a negative effect on Type-GBFS when combined with either Preferred Operators or Deferred Evaluation, but work very well when combined with both. Finding an explanation for this surprising behavior is left as future work. The last row in Table 2 lists coverage results when all three enhancements are applied as in LAMA.

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

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).
- **Type-LAMA:** LAMA-2011 with GBFS replaced by Type-GBFS, uses the same four queues as LAMA-2011, plus (h^{FF}, g) type buckets.
- **DBFS2:** DBFS2 (Imai and Kishimoto 2011), an enhanced version of DBFS which adds a second global open list for preferred operators only, was re-implemented by its original author on the LAMA-2011 code base, which is much stronger than the LAMA-2008 code base used in (Imai and Kishimoto 2011). The parameters $P = 0.1, T = 0.5$ from the same paper are used.

LAMA-2011 and Type-LAMA correspond to PO+DE+MH in Table 2.

domain	size	LAMA-2011	Type-LAMA	DBFS2
98-logistics	35	**35**	**35**	34
98-mystery	30	**19**	**19**	**19**
00-miconic-full	150	136	**139**	138.6
02-depot	22	20	**21.6**	18.2
02-freecell	80	78	77.8	**79.8**
04-airport-strips	50	32	34.6	**41**
04-notankage	50	**44**	**44**	43.2
04-optical-tele	48	4	**5.4**	5
04-philosopher	48	39	**48**	**48**
04-psr-large	50	**32**	31.6	15.6
04-satellite	36	**36**	35.2	27
06-pathways	30	**30**	**30**	28.4
06-storage	30	18	**23.8**	23.4
06-tankage	50	41	**42**	36.8
06-trucks-strips	30	14	20.8	**24**
08-scanalyzer	30	**30**	**30**	29.6
08-sokoban	30	**28**	27	**28**
08-transport	30	29	**30**	29.8
08-woodworking	30	**30**	29.8	**30**
11-elevators	20	**20**	**20**	18.8
11-floortile	20	6	5.6	**7.6**
11-nomystery	20	10	**17.8**	**17.8**
11-openstacks	20	**20**	**20**	12.8
11-parking	20	**18**	**17.6**	12.6
11-scanalyzer	20	**20**	**20**	19.8
11-sokoban	20	**19**	18.2	18
11-tidybot	20	16	**16.4**	16.2
11-transport	20	**16**	15.6	12.4
11-visitall	20	**20**	**20**	7
Total	2112	1913	**1949.8**	1895.4
Unsolved	-	199	**162.2**	216.6

Table 3: Number of instances solved. 25 domains with 100% coverage for all three planners omitted.

Table 3 shows detailed coverage differences of these three planners. Type-LAMA outperforms the other two planners, solving 36.8 more problems than LAMA-2011 and 54.4 more than DBFS2. The percentage of unsolved problems is reduced from 9.4% for LAMA to 7.6%. This is comparable to the improvement from adding the *Landmark count* heuristic (Richter and Westphal 2010) - from 11.4% to 9.4% unsolved. Considering that only the hardest problems were left unsolved, adding the type system makes the planner substantially stronger.

Figures 3(b) and (c) compare the search time of Type-LAMA against LAMA-2011 and DBFS2 in the manner described for Figure 3(a). Between Type-LAMA and LAMA-2011, many results are very close, presumably for instances where exploration plays only a small role. Type-LAMA has a large time advantage of over $10\times$ more often. Results for Type-LAMA and DBFS2 are much more diverse. Besides its coverage advantage, Type-LAMA also wins the time comparison for a large number of instances by factors of over $2\times$, $10\times$ and $50\times$.

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

Although type buckets cause some change in solution costs, the influence is not clear-cut. If we compare LAMA-2011 and one typical run Type-LAMA's results, there are 1907 problems solved by both planners. The IPC scores for LAMA-2011 and Type-LAMA are 1895.1 vs 1892.5, only 2.6 difference over the 1907 problems. For 1698 problems, both planners get the same cost solutions.

Effect of Different Type Systems

The results above for both Type-GBFS and Type-LAMA are for the (h^{FF}, g) type system. Table 4 summarizes results for these two planners when using several other simple type systems. (1) is the trivial single-type system $T(s) = 1$.

Among single-element type systems, (g) performs better than either heuristic, and (h^{lm}) solves around 10 more problems than (h^{FF}). Since Type-GBFS only uses h^{FF}, h^{lm} is only tested for Type-LAMA.

Compared to GBFS, (g) explores much more on nodes with low g-values, typically at shallow levels of the search tree. Many such nodes will be ignored or expanded very late in GBFS. In contrast, an (h)-only type system focuses on exploring different estimated goal distances and ignores g. Interestingly, (g) is even slightly better than (h^{FF}, g) in Type-GBFS, but (h^{FF}, g) is better in Type-LAMA. Among two-element type systems, (h^{FF}, g) and (h^{lm}, g) are the top two configurations, while (h^{FF}, h^{lm}) is just slightly better than (h^{lm}). The three-element type system (h^{FF}, h^{lm}, g) is worse and might be too fine-grained for this test set. The question of the right granularity is important and needs further study.

Type	T-GBFS	T-LAMA	Type	T-LAMA
none	1561	1913	(h^{lm})	1921.6
(1)	1529.6	1916.2	(h^{lm}, g)	1942.4
(g)	**1758.2**	1935.0	(h^{FF}, h^{lm})	1925.6
(h^{FF})	1729.0	1918.6	(h^{FF}, h^{lm}, g)	1939.0
(h^{FF}, g)	1755.6	**1949.8**		

Table 4: Coverage of Type-GBFS (T-GBFS) and Type-LAMA (T-LAMA) with simple type systems.

Type-LAMA Works Better as an Integrated System than as a Simple Portfolio

This experiment compares Type-LAMA, which integrates type-based exploration directly into LAMA's search process, with a portfolio which independently runs LAMA and a simple type-based planner ST. ST selects nodes exclusively from type buckets, using a (h^{FF}, g) type system as defined above. For consistency with LAMA, Deferred Evaluation is used for type buckets as well.

By itself, ST solves 1266 out of 2112 IPC problems. Consider a portfolio planner that uses x seconds for LAMA-2011, followed by $1800 - x$ seconds for ST. The best coverage of 1926 is achieved for $x = 1279$. Type-LAMA solves 23.8 problems more than this best portfolio. This shows the synergy between exploitation-based search in LAMA and exploration using a type system.

Conclusion and Future Work

The primary contributions of this paper are an investigation of the problem of inefficient exploration in previous GBFS-type planners, and the solution of using type-based exploration. The new algorithm Type-GBFS samples nodes uniformly over a type system instead of uniformly over all nodes in an open list. By replacing GBFS with Type-GBFS, the planner Type-LAMA can solve 36.8 more problems than LAMA-2011 on average, decreasing the number of unsolved problems over all past IPC domains from 199 to 162.2.

One obvious direction for future work is to test many more type systems, including others proposed by Lelis, Zilles and Holte (2013). Different forms of boosting and non-uniform exploration of different types could also be investigated. Regarding experimental results, it is unclear why the combination of multiple heuristics with either one of preferred operators and deferred evaluation fails, and yet succeeds when combined with both.

One potential problem of Type-LAMA is that the type system might not be able to explore deeply enough when the distance from current nodes in the open list to heuristically promising nodes is large. One potential solution for this problem is to use a larger local search (Imai and Kishimoto 2011; Xie, Müller, and Holte 2014), or other forms of exploration such as random walks (Nakhost and Müller 2009).

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, AITF, Alberta Innovates Technology Futures, and Grant-in-Aid for JSPS Fellows, Japan Society for the Promotion of Science.

References

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

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

Felner, A.; Kraus, S.; and Korf, R. E. 2003. KBFS: K-best-first search. *Ann. Math. Artif. Intell.* 39(1-2):19–39.

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

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.

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., *Proceeding of the 12th International Conference on Autonomous Agents and Multi-Agent Systems*, 555–562.

Nakhost, H., and Müller, M. 2009. Monte-Carlo exploration for deterministic planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-2011)*, 1766–1771.

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 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)*, 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)*.

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)*.