A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning

Tatsuya Imai
Tokyo Institute of Technology

Akihiro Kishimoto
Tokyo Institute of Technology and JST PRESTO

Abstract

Greedy best-first search (GBFS) is a popular and effective algorithm in satisficing planning and is incorporated into high-performance planners. GBFS in planning decides its search direction with automatically generated heuristic functions. However, if the heuristic functions evaluate nodes inaccurately, GBFS may be misled into a valueless search direction, thus resulting in performance degradation. This paper presents a simple but effective algorithm considering a diversity of search directions to avoid the errors of heuristic information. Experimental results in solving a variety of planning problems show that our approach is successful.

Introduction

In satisficing planning, where suboptimal solutions are accepted, many planners employ best-first search strategies including greedy best-first search (GBFS) (e.g., (Bonet and Geffner 2001; Helmert 2006)). Let h be a heuristic function that estimates the distance to a goal from a node n. GBFS selects the best node n with the smallest $h(n)$ in the open list that maintains nodes that have been generated but have not been expanded yet. It then expands n to generate n's successors, and saves these successors in the open list, unless they have been previously added to the open list. Next, it saves n in the closed list that keeps the nodes that have been expanded. It continues these steps until finding a goal node or proving that there is no solution in the search space.

Heuristic functions play an important role in drastically improving performance of GBFS. While automatic generation of heuristic functions (e.g., (Hoffmann and Nebel 2001; Helmert 2006)) enables state-of-the-art satisficing planners to solve very complicated planning problems including benchmarks in the International Planning Competitions, accurate evaluations of nodes still remain as a challenging task.

Although GBFS is fundamental and powerful in planning, it has an essential drawback when heuristic functions return inaccurate estimates. Assume that a heuristic function underestimates the difficulties of unpromising nodes. Then, since GBFS must expand nodes with small heuristic values first, it spends most of time in searching only unpromising areas and delays moving to the promising part. Figure 1 illustrates a typical transition of heuristic values of nodes selected for expansions when GBFS with the FF heuristic (Hoffmann and Nebel 2001) solves a planning problem. The horizontal axis indicates each expansion of the best node n in the open list and the vertical axis represents n's corresponding heuristic value for that expansion. Circles, the triangle, and diamond represent expanding nodes that are on the path to a goal. This transition indicates that until reaching a node m marked by diamond GBFS keeps expanding many valueless nodes erroneously evaluated as more promising than m by the heuristic function, after expanding a node marked by triangle. These valueless nodes never contribute to solving the problem.

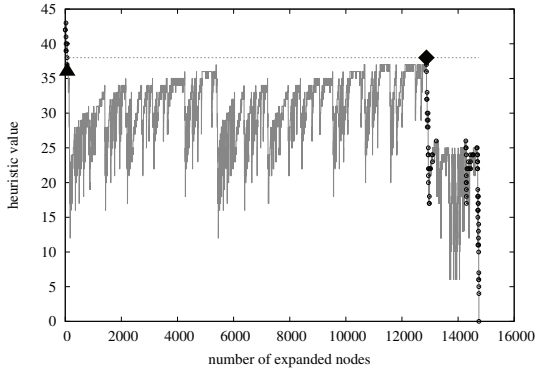

Figure 1: A transition of heuristic values in solving opticaltelegraphs #02

Previous work tackles this issue by adding a *diversity* to search, which is an ability in simultaneously exploring different parts of the search space to bypass large errors in heuristic functions. For example, several algorithms combined with diversity such as in (Felner, Kraus, and Korf 2003; Linares López and Borrajo 2010; Röger and Helmert 2010) are empirically shown to be superior to naive best-first search algorithms. However, they still have limited diversity, since they do not immediately expand nodes mistakenly evaluated as very unpromising ones.

This paper presents a new technique that incorporates a diversity into search in a different way than previous search-based approaches. Our contributions are summarized as:

1. Diverse best-first search (DBFS) that is robust to large heuristic evaluation errors. Even if a heuristic function

erroneously evaluates promising node n as unpromising, DBFS can occasionally expand n. The frequency of selecting such n is controlled by the heuristic value of n and the path cost to n from the root node.

2. Empirical results clearly showing that DBFS is effective in satisficing planning. DBFS outperforms GBFS and a previous approach in (Felner, Kraus, and Korf 2003). Additionally, by combining with a popular enhancement technique, DBFS solves more planning instances than the Fast Downward planner (Helmert 2006).

Related Work

Although similar issues have been addressed in other domains such as balancing exploration and exploitation (Kocsis and Szepesvári 2006) and escaping from local minima (Selman, Levesque, and Mitchell 1992), we review the literature on planning and diversifying search directions.

K-best-first search (KBFS(k)) is a generalization of best-first search (Felner, Kraus, and Korf 2003). KBFS(k) expands the k best nodes in the open list at a time, and then inserts the successors of the k nodes into the open list. It repeats this process until finding a goal node or proving no solution. This algorithm provides a simple way of adding a diversity to search by delayed examinations of successors. Parameter k controls the diversity; a more variety of paths are considered for larger k. KBFS(k) and an extended version of KBFS(k) (EKBFS(k)) were applied to classical planning in (Linares López and Borrajo 2010). EKBFS incorporates a few enhancements introduced in (Koehler and Hoffmann 2000; Hoffmann and Nebel 2001). Linares López and Borrajo conclude that in solving hard problems both KBFS(k) and EKBFS(k) outperform GBFS and enhanced hill climbing presented in (Hoffmann and Nebel 2001).

KBFS(k) partially avoids search plateaus caused by misleading heuristic estimates. However, if the heuristic values of all the k best nodes are erroneously underestimated, it results in searching useless areas. Although the possibility of occurring this drawback can be reduced by increasing k, KBFS(k) tends to behave similarly to breadth-first search, thus losing the benefit from the heuristic information.

The alternation method uses more than one heuristic function (Helmert 2006; Röger and Helmert 2010). It manages an open list for each heuristic function, and selects one of the open lists in a round-robin manner in each node expansion. Alternation diversifies search directions by expecting heuristic functions to evaluate nodes differently. However, if all of them inaccurately evaluate unpromising nodes as promising, it suffers from an excessive overhead of expanding valueless nodes. Dovetailing is based on a more general idea than alternation, since it runs different search algorithms such as weighted A* with various weights (Valenzano et al. 2010). However, it essentially has a similar dilemma to alternation.

Other approaches adding a diversity with an application to planning include a restarting procedure combined with local search (Coles, Fox, and Smith 2007) and random walk (Nakhost and Müller 2009). While their approaches avoid plateaus caused by misleading heuristic estimates by almost completely forgetting the previously explored search space, they may suffer from duplicate search effort such as re-expanding the same nodes via different paths many times.

The Diverse Best-First Search Algorithm

Our DBFS algorithm overcomes issues addressed in the last section. As in approaches with restarting procedures (e.g., (Nakhost and Müller 2009)), DBFS diversifies search directions by probabilistically selecting a node that does not have the best heuristic value. As a result, compared to KBFS(k) and weighted A*, DBFS has a higher chance of expanding a node mistakenly estimated to be unpromising. Additionally, unlike the approaches with restarting, DBFS performs more systematic search by keeping all the expanded nodes in the closed list. It can therefore effectively reuse search results such as the case of which there are many paths to the same node. Moreover, with heuristic function h, if $h(n) \neq \infty$ holds for any node n that is reachable to a goal, DBFS is a complete algorithm. In other words, with infinite time and memory, it can return a solution when a solution exists, and terminate correctly when there is no solution.

Algorithms 1 and 2 show the pseudo-code of DBFS. The framework of DBFS is very simple. Until finding a goal node or proving no solution, it repeats the procedures of fetching one node n from the global open list (OL in the pseudo-code) and performing GBFS rooted at n with the local open list (LocOL in the pseudo-code). DBFS optimistically expects GBFS to find a solution for n with the smallest search effort. The number of nodes expanded per GBFS is therefore limited to $h(n)$, which is the minimum number of nodes that must be expanded to find a goal with the unit edge cost if $h(n)$ does not overestimate the distance to the goal. Tied heuristic values are broken randomly in GBFS. As similarly presented in (Botea and Ciré 2009), duplicate search effort is eliminated by the shared global closed list. After GBFS expands $h(n)$ nodes, all the nodes in the local open list are inserted to the global open list to make these nodes as candidates for a selection in the next node-fetching phase.

Algorithm 1 Diverse Best-First Search

```
 1: insert the root node into OL;
 2: while OL is not empty do
 3:     n := fetch a node from OL;
 4:     LocOL := {n};
 5:     /* Perform GBFS rooted at n */
 6:     for i:=1 to h(n) do
 7:         select node m with the smallest h(m) from LocOL;
 8:         if m is a goal then
 9:             return plan to m from the root;
10:         end if
11:         save m in the global closed list;
12:         expand m;
13:         save m's successors in LocOL;
14:     end for
15:     OL: = LocOL ∪ OL;
16: end while
17: return no solution;
```

Algorithm 2 presents the procedure of fetching a node, which is called at line 3 of Algorithm 1. Let $g(n)$ be the

Algorithm 2 Fetching one node

```
1: p_total := 0;
2: (h_min, h_max) := minimum and maximum h-values in OL;
3: (g_min, g_max) := minimum and maximum g-values in OL;
4: if with probability of P then
5:     G := select at random from g_min, ··· , g_max;
6: else
7:     G := g_max;
8: end if
9: for all h ∈ {h_min, ··· , h_max} do
10:    for all g ∈ {g_min, ··· , g_max} do
11:        if g > G or OL has no node whose h-value and g-value
           are h and g, respectively then
12:            p[h][g] := 0;
13:        else
14:            p[h][g] := T^(h−h_min);
15:        end if
16:        p_total := p_total + p[h][g];
17:    end for
18: end for
19: select a pair of h and g with probability of p[h][g]/p_total;
20: dequeue a node n with h(n) = h and g(n) = g in OL;
21: return n;
```

g-value of node n, which is the sum of the edge costs on the path from the root node to n, and $h(n)$ be the heuristic value (h-value in short) of n. A node n selected to perform GBFS is determined by a probability computed by $h(n)$ and $g(n)$. If more than one node has the same pair of h and g values, one of them is chosen randomly.

Parameters P and T ($0 \leq P, T \leq 1$) decide a policy of fetching the next node. When GBFS is used for global search, it tends to select nodes with large g-values due to greediness of repeatedly selecting a successor that appears to be promising. P enables DBFS to restart exploring the search space that is closer to the root, where DBFS has not yet exploited enough to find the promising nodes. On the other hand, T controls the frequency of selecting a node n based on the gap between the current best h-value and $h(n)$. Lower probabilities are assigned to nodes with larger h-values to balance exploiting the promising search space and exploring the unpromising part. Heuristic estimates are completely ignored if $T = 1$. On the other hand, DBFS fetches the same node chosen by GBFS if $T = 0$ and $P = 0$.

If GBFS selects an unpromising node n and n's descendants have smaller h-values than $h(n)$, it keeps saving n's unpromising descendants in the open list and expanding them. However, even if DBFS fetches n, it expands $h(n)$ nodes and then selects another node that may not be n's descendant. Only at most $b \cdot h(n)$ nodes are inserted to OL where b is the largest number of edges of the $h(n)$ nodes. This number is much smaller than that of GBFS, since GBFS must store all the useless descendants.

Experimental Results

Setup

The performance of DBFS was evaluated by running experiments on solving 1,612 planning instances in 32 domains from the past five International Planning Competitions. We built all the implementations on top of the Fast Downward planner (Helmert 2006). All the experiments were run on a dual quad-core 2.33 GHz Xeon E5410 machine with 6 MB L2 cache. The time and memory limits for solving an instance were set to 30 minutes and 2 GB. We observed both cases of which all the evaluated algorithms ran out of memory and exceeded the time limit when they were unable to solve instances. We excluded the translation time from the PDDL representation (Edelkamp and Hoffmann 2004) into the SAS+ representation (Bäckström and Nebel 1995), which was preprocessed by Fast Downward.

Performance Comparisons without Enhancements

First, we analyze strengths and weaknesses of DBFS, GBFS and KBFS by disabling enhancements in Fast Downward (e.g., preferred operators and multiple heuristic functions), thus measuring the potential of each algorithm. We used the FF (Hoffmann and Nebel 2001), causal graph (CG) (Helmert 2006) and context-enhanced additive (CEA) (Helmert and Geffner 2008) heuristics already implemented in Fast Downward. The best known random seed was used for each heuristic function. However, DBFS solved all the instances with the same seed and with $P = 0.1$ and $T = 0.5$, and did not exploit the best seed for each instance.

Table 1 shows the number of solved instances when all the algorithms used the FF heuristic. We assumed that KBFS(k) could exploit the best k for each planning domain. We therefore ran KBFS(2^l) for all the cases of integer l satisfying $0 \leq l \leq 7$, and included the best result for each domain.

Table 1 clearly indicates the superiority of DBFS to KBFS and GBFS. DBFS either solved an equal or larger number of instances than the others in all the domains. In particular, DBFS performed much better in the Schedule domain. Of 150 instances, DBFS solved 129 problems while GBFS and KBFS solved only 18 and 46 instances, respectively. However, even if we excluded this domain, DBFS was still able to solve at least 80 additional instances in total compared with the other approaches. Hence, our results imply the importance of diversifying search directions.

KBFS solved additional instances in several domains compared to GBFS such as Airport, Assembly, Pathways and Schedule. However, KBFS usually achieved smaller performance improvements than DBFS. Additionally, we observed that selecting the best k in KBFS(k) played an important role in improving its solving ability, although selecting such k automatically remains an open question. For example, in the Philosophers domain, while KBFS(1) (i.e., identical to GBFS) solved all 48 instances, KBFS(128) was able to solve only 25 instances.

Figure 2 compares the number of nodes expanded by GBFS and DBFS with the FF heuristic for the instances solved by both. The node expansion of GBFS was plotted on the horizontal axis against DBFS on the vertical axis on logarithmic scales. A point below the linear line indicates that DBFS expanded fewer nodes than GBFS in solving one instance. Figure 2 clearly shows that DBFS outperformed GBFS especially when solving hard instances. Of 1,208 instances solved by both, it took either DBFS or GBFS at least one second to solve each of 279 instances. Of these

Table 1: The number of instances solved by each algorithm with the FF heuristic and without enhancements

Domain	GBFS	KBFS	DBFS
Airport (50)	33	44	**46**
Assembly (30)	18	27	**30**
Blocks (35)	**35**	**35**	**35**
Depot (22)	16	17	**19**
Driverlog (20)	18	**20**	**20**
Freecell (80)	**80**	**80**	**80**
Grid (5)	**5**	**5**	**5**
Gripper (20)	**20**	**20**	**20**
Logistics 1998 (35)	30	31	**33**
Logistics 2000 (28)	**28**	**28**	**28**
Miconic (150)	**150**	**150**	**150**
Miconic Full ADL (150)	135	137	**139**
Miconic Simple ADL (150)	**150**	**150**	**150**
Movie (30)	**30**	**30**	**30**
MPrime (35)	26	27	**33**
Mystery (30)	16	17	**19**
Openstacks (30)	28	28	**30**
Optical Telegraphs (48)	3	3	**5**
Pathways (30)	9	16	**30**
Philosophers (48)	**48**	**48**	**48**
Pipesworld Notankage (50)	31	37	**44**
Pipesworld Tankage (50)	24	25	**35**
PSR Large (50)	31	31	**32**
PSR Middle (50)	**50**	**50**	**50**
PSR Small (50)	**50**	**50**	**50**
Rovers (40)	27	28	**37**
Satellite (36)	25	26	**28**
Schedule (150)	18	46	**129**
Storage (30)	19	21	**25**
TPP (30)	22	23	**29**
Trucks (30)	14	18	**22**
Zenotravel (20)	**20**	**20**	**20**
Total (1612)	1,209	1,288	**1,451**

279 instances, DBFS expanded fewer nodes than GBFS in solving 209 instances. This resulted in a large difference in search time (see Figure 3 comparing the search time with the FF heuristic for the instances solved by both). DBFS solved 194 instances more quickly and was five times faster than GBFS in solving the aforementioned 279 instances. The overhead of DBFS fetching a node did not offset its benefit of achieving drastic reductions of node expansions. In fact, because most computational overhead incurred by DBFS was not the node-fetching phase but the procedure of performing GBFS, the node expansion rate of DBFS was similar to that of GBFS. The figures showing performance comparisons between DBFS and KBFS are omitted, since we obtained similar results.

Figure 4 compares the quality of plans (i.e., solution lengths) computed by DBFS and GBFS with the FF heuristic. DBFS often returned longer solutions than GBFS, since DBFS selected unpromising nodes that tend to be on a more redundant path to a goal. This phenomenon was similarly observed in (Nakhost and Müller 2009), when their planner was compared against Fast Downward in a few domains. However, since DBFS still yielded similar plans in many

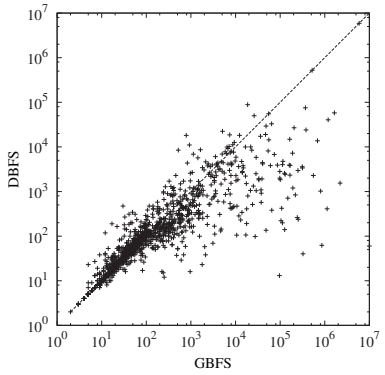

Figure 2: Comparison of node expansions between GBFS and DBFS with the FF heuristic

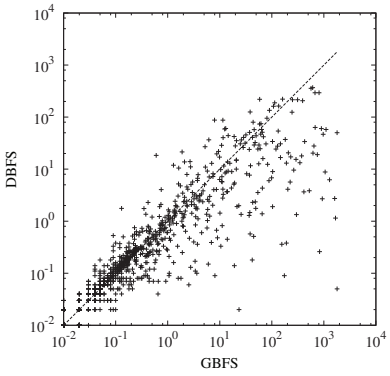

Figure 3: Search time for instances solved by DBFS and GBFS with the FF heuristic

cases, this is a price to pay for achieving performance improvements.

Table 2: The number of instances solved by each algorithm with the CG/CEA heuristic and without enhancements

Heuristic	GBFS	KBFS	DBFS
CG	1,170	1,218	**1,358**
CEA	1,202	1,240	**1,388**

Table 2 shows the total number of solved instances in all domains with the CG or CEA heuristic. Numbers are calculated in the same way as in Table 1. The superiority of DBFS was confirmed even with different heuristics. DBFS performed worse than the others only in a few domains.

Performance Comparisons with Various Parameters and Random Seeds

Next, we varied parameters P and T in the range of $0.1-0.3$ and $0.4-0.6$, respectively, in increments of 0.1 (i.e., we examined nine combinations of parameters for each heuristic function). Table 3 shows the average, minimum and maximum numbers of solved instances. The same random seed

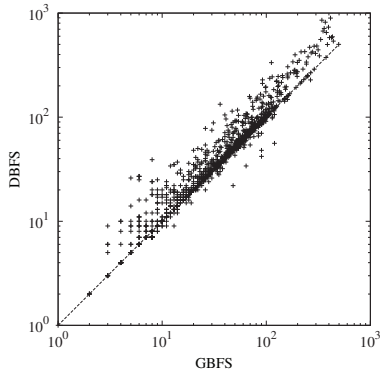

Figure 4: Comparison of plan lengths for instances solved by GBFS and DBFS with the FF heuristic

Table 3: Performance of DBFS with different parameters

Heuristic	Average	Minimum	Maximum
FF	1,438	1,432	1,451
CG	1,345	1,335	1,361
CEA	1,370	1,358	1,388

was used for each pair of parameters. Results show that DBFS outperformed both GBFS and KBFS by a large margin with any of the three heuristic functions and even with the worst parameter settings. Additionally, DBFS was robust to the changes of P and T. While the differences between the minimum and maximum numbers of solved instances were 30 with the CEA heuristic, DBFS was still able to solve most of the instances. While the best value of T depended on heuristic functions and P, we observed that performance tended to deteriorate with a larger value of P. All of the worst case scenarios shown in Table 3 were obtained with $P = 0.3$.

Table 4: Performance of DBFS with resetting one parameter

Heuristic	**FF**	**CG**	**CEA**
$P = 0.1$ (and $T = 0$)	1,366	1,299	1,309
$P = 0.2$	1,362	1,296	1,306
$P = 0.3$	1,358	1,278	1,313
$T = 0.4$ (and $P = 0$)	1,422	1,338	1,381
$T = 0.5$	1,432	1,336	1,377
$T = 0.6$	1,433	1,321	1,364

Table 4 shows number of solved instances when either P or T is fixed to zero and the other parameter is varied to show the behavior of DBFS with extremely ineffective parameter settings. No heuristic information is used to diversify search directions with $T = 0$, while only h-values are considered for diversity with $P = 0$. It is not surprising to observe performance degradation compared to the case where P and T are non-zero, since valuable information (i.e., h-values or g-values) is unused. However, DBFS still outperformed GBFS and KBFS with all experimented parameter settings, clearly indicating the importance of escaping from search plateaus.

Table 5: Performance of DBFS with different random seeds

Heuristic	Average	Minimum	Maximum
FF	1,447	1,443	1,451
CG	1,353	1,349	1,358
CEA	1,381	1,377	1,388

Table 5 shows the performance of DBFS with different random seeds for each heuristic. We examined five seeds with fixed parameters $P = 0.1$ and $T = 0.5$. This table clearly shows that DBFS was robust to the change of random seeds. Almost all the instances remained solvable even if we changed the seeds. For example, only 11 instances became unsolvable when the best seed was changed to the worst one with the CEA heuristic.

Performance Comparisons with an Enhancement

Next, the performance of each algorithm was evaluated with turning on enhancements. Table 6 shows the number of instances solved by the following algorithms:

EKBFS : KBFS(k) with the FF heuristic, enhanced with preferred operators (Helmert 2006) (a.k.a. helpful actions in (Hoffmann and Nebel 2001)). As in (Linares López and Borrajo 2010) and Fast Forward, our current EKBFS implementation first expands only preferred successors, and then performs the KBFS(k) search for the other successors. However, unlike in (Linares López and Borrajo 2010), goal agenda was not incorporated, because it was not implemented in Fast Downward. Additionally, as we did in the previous subsection, after running KBFS(2^l) for all the cases of integer l satisfying $0 \leq l \leq 7$, we calculated the total number based on the best result in each domain.

FD : The state-of-the-art Fast Downward planner with four enhancements (alternation based on the FF and CEA heuristics, deferred evaluation, preferred operators and boosting (Helmert 2006; Richter and Helmert 2009)). In the preliminary experiments, we tried all combinations of alternation among the FF, CEA and CG heuristics, and the other three enhancements, and chose the configuration with the best solving ability.

DBFS2 : DBFS with the FF heuristic, enhanced with preferred operators and modified as follows: An additional global open list was prepared separately for preferred successors. After the node-fetching algorithm selects a node randomly from one of the global open lists, DBFS2 performs GBFS rooted at that node. We used 0.1 for parameter P and 0.5 for T.

Table 6: The number of instances solved by each algorithm with turning on enhancements

	EKBFS	**FD**	**DBFS2**
Total (1612)	1,382	1,458	**1,481**

Despite a smaller number of enhancements currently incorporated into DBFS2 than FD, DBFS2 solved the largest number of instances, showing the superiority of our approach. The performance difference between DBFS2 and EKBFS became smaller than in Table 1. This was mainly due to the increased number of instances solved in the Schedule domain. With the help of preferred operators, EKBFS solved 142 instances in this domain, while KBFS did only 46 of 150 instances. However, DBFS2 still outperformed EKBFS by a large margin. It seems to exploit the promising search space that is orthogonal to preferred operators.

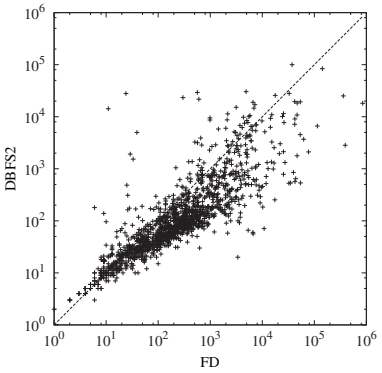

Figure 5: Comparison of node expansions between FD and DBFS2

Figure 5 compares node expansions solved by both FD and DBFS2. DBFS2 drastically reduced node expansions compared to FD. Of 1,445 instances solved by both, DBFS2 expanded fewer nodes than FD in solving 1,106 instances and was 1.5 times faster in solving the 1,445 instances.

Figure 6 shows a comparison of plan lengths between FD and DBFS2. Compared to Figure 4, we observed a smaller

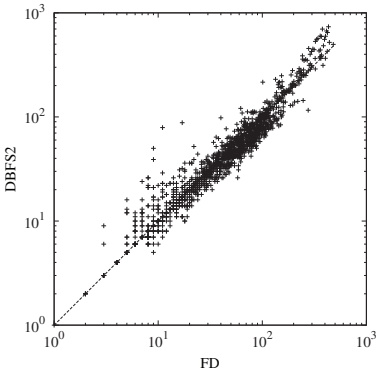

Figure 6: Comparison of plan lengths between FD and DBFS2

difference in the quality of plans, because preferred operators contributed to improving the plan quality of DBFS2 and Fast Downward often returned longer plans than GBFS.

Performance Comparison to LAMA

The LAMA planner is a variant of Fast Downward and returns the better quality of plans by first performing greedy best-first search and then refining plans with a series of weighted A* (WA*) search that gradually decreases weight values (Richter and Westphal 2010) until it reaches a time limit. The landmark and FF heuristics are used in LAMA with various enhancements similar to Fast Downward. One way to combine our approach with LAMA is to replace the first phase of greedy best-first search by DBFS2.

Since the first search phase determines the solving ability, LAMA with DBFS2 solved 1,481 instances as in Table 6. On the other hand, LAMA solved 1,445 instances[1].

Figure 7 compares plan lengths for the instances solved by LAMA and LAMA with DBFS2. The plan quality was

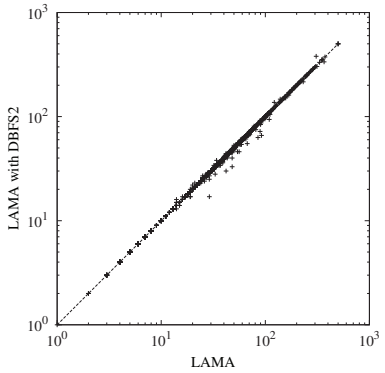

Figure 7: Comparison of plan lengths between LAMA and LAMA with DBFS2

mostly similar between these methods. Of 1,438 instances solved by both, LAMA with DBFS2 returned plans with the same lengths as LAMA in 1,310 instances. This indicates that plans can be later refined by LAMA's WA* search while DBFS2 can improve its solving ability.

Conclusions and Future Work

This paper described the DBFS algorithm avoiding plateaus of search caused by misled heuristic estimates. Experimental results showed that DBFS outperformed GBFS and KBFS(k) in satisficing planning as well as was robust to the changes of parameters and random seeds. By incorporating preferred operators, DBFS performed better than the Fast Downward planner. Additionally, by combining DBFS with the LAMA planner, our approach not only improved the solving ability of LAMA but also returned plans with reasonable quality. We therefore conclude that DBFS can be a strong candidate as a new baseline in satisficing planning.

There are several ideas to strengthen DBFS. One is to develop a better node-fetching method by incorporating

[1] A main culprit obtaining a smaller number than Fast Downward would be due to a difference in heuristics between LAMA and Fast Downward (landmark versus CEA). Although this number could be increased to 1,458 by replacing the first search phase by Fast Downward, LAMA with DBFS2 still performed better.

state-of-the-art techniques elegantly restarting search such as (Nakhost and Müller 2009). We believe that they can be transposed to DBFS. Another is to combine DBFS with some enhancements specific to satisficing planning. In the current implementation, DBFS is combined with only preferred operators. Synthesizing DBFS with other enhancements such as goal agenda (Koehler and Hoffmann 2000), boosting (Richter and Helmert 2009), and multiple heuristic functions would exploit the more promising search space in an orthogonal way to DBFS. Additionally, since DBFS is a general search algorithm, it should not be limited to planning in principle. Applying DBFS to other domains is therefore of interest and value as future work. Finally, it is important to analyze the behavior of DBFS both theoretically and empirically. For example, although DBFS achieves drastic improvements in the Schedule domain, we have not yet had a strong ground to explain why it performs much better than GBFS. We are currently trying to develop a theoretical model to analyze the strengths and weaknesses of DBFS.

Acknowledgments

We would like to thank Adi Botea, Alex Fukunaga, Hootan Nakhost and Martin Müller for their comments on the paper. This research is supported by the JST PRESTO program.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129:5–33.

Botea, A., and Ciré, A. A. 2009. Incremental heuristic search for planning with temporally extended goals and uncontrollable events. In *Proceedings of IJCAI 2009*, 1647–1652.

Coles, A.; Fox, M.; and Smith, A. 2007. A new local-search algorithm for forward-chaining planning. In *Proceedings of ICAPS 2007*, 89–96.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th International Planning Competition. Technical report, Albert-Ludwigs-Universität Freiburg, Institute für Informatik.

Felner, A.; Kraus, S.; and Korf, R. E. 2003. KBFS: K-best-first search. In *Annals of Mathematics and Artificial Intelligence*, volume 39, 19–39.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In *Proceedings of ICAPS 2008*, 140–147.

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.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, volume 4212 of *Lecture Notes in Computer Science*, 282–293. Springer.

Koehler, J., and Hoffmann, J. 2000. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. 12:339–386.

Linares López, C., and Borrajo, D. 2010. Adding diversity to classical heuristic planning. In *Proceedings of the Third Annual Symposium on Combinatorial Search (SoCS-10)*, 73–80.

Nakhost, H., and Müller, M. 2009. Monte-Carlo exploration for deterministic planning. In *Proceedings of IJCAI 2009*, 1766–1771.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of 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.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In *Proceedings of ICAPS 2010*, 246–249.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In *Proceedings of AAAI 1992*, 440–446.

Valenzano, R.; Sturtevant, N.; Schaeffer, J.; Buro, K.; and Kishimoto, A. 2010. Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms. In *Proceedings of ICAPS 2010*, 177–184.