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

Introduction

Heuristic functions play an important role in drastically improving performance of satisficing planners based on greedy best-first search (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.

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. Several algorithms combined with diversity (e.g., K-best-first search (KBFS) in (Felner, Kraus, and Korf 2003)) 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 called *diverse best-first search (DBFS)*, which incorporates a diversity into search in a different way than previous search-based approaches. We show empirical results clearly showing that DBFS is effective in satisficing planning.

The Diverse Best-First Search Algorithm

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 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). The number of nodes expanded per GBFS is limited to $h(n)$. That is, even if DBFS fetches unpromising node n, it expands only $h(n)$ of n's descendants and then selects another node that may not be n's descendant. 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 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 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.

Algorithm 2 Fetching one node

```
p_total := 0;
(h_min, h_max) := minimum and maximum h-values in OL;
(g_min, g_max) := minimum and maximum g-values in OL;
if with probability of P then
    G := select at random from g_min, ..., g_max;
else
    G := g_max;
end if
for all h ∈ {h_min, ..., h_max} do
    for all g ∈ {g_min, ..., g_max} do
        if g > G or OL has no node whose h-value and g-value
        are h and g, respectively then
            p[h][g] := 0;
        else
            p[h][g] := T^(h−h_min);
        end if
        p_total := p_total + p[h][g];
    end for
end for
select a pair of h and g with probability of p[h][g]/p_total;
dequeue a node n with h(n) = h and g(n) = g in OL;
return n;
```

Experiments

We evaluated the performance of DBFS, GBFS and KBFS. 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 built all the implementations on top of the Fast Downward planner (Helmert 2006). We ran these three search algorithms with either the FF (Hoffmann and Nebel 2001), causal graph (CG) (Helmert 2006), or context-enhanced additive (CEA) (Helmert and Geffner 2008) heuristic. The other enhancements in Fast Downward were turned off. We used 1,612 planning instances in 32 domains from the past five International Planning Competitions for the experiments. We ran DBFS with same parameters $P = 0.1$ and $T = 0.5$ and same random seed for all the instances. Although we included the best result in the paper after measuring the performance with several random seeds and various P and T, DBFS was very robust to the changes of these parameters. On the other hand, We ran KBFS(2^l) for all the cases of integer l satisfying $0 \leq l \leq 7$, and included the best result for each problem.

Table 1: The number of instances solved by each algorithm

Heuristic	**GBFS**	**KBFS**	**DBFS**
FF	1,209	1,288	**1,451**
CG	1,170	1,218	**1,358**
CEA	1,202	1,240	**1,388**

Table 1 shows the number of instances solved by each algorithm. Table 1 clearly indicates the superiority of DBFS to KBFS and GBFS. DBFS solved the largest number of instances with all heuristic functions. In particular, DBFS either solved an equal or larger number of instances than the others in all the domains with the FF heuristic. KBFS solved additional instances in several domains compared to GBFS. However, KBFS usually achieved smaller performance improvements than DBFS.

Figure 1 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 1 clearly shows that DBFS outperformed

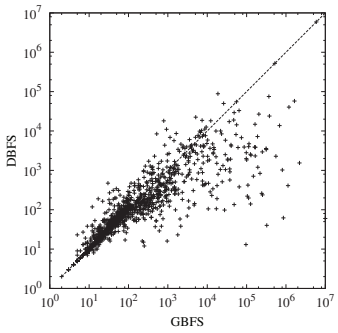

Figure 1: Comparison of node expansions between GBFS and DBFS with the FF heuristic

GBFS especially when solving hard instances.

DBFS often returned longer solutions than GBFS, since DBFS selected unpromising nodes that tend to be on a more redundant path to a goal. However, since DBFS still yielded similar plans in many cases, this is a price to pay for achieving performance improvements.

We also evaluated the performance of DBFS with the FF heuristic and preferred operators, against the state-of-the-art Fast Downward (FD) planner with the best combination of enhancements (alternation based on the FF and CEA heuristics, deferred evaluation, preferred operators and boosting (Helmert 2006; Richter and Helmert 2009)). A smaller number of enhancements was incorporated into DBFS than FD. However, while FD solved only 1,458 instances, DBFS solved 1,481 instances.

References

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.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of ICAPS 2009*, 273–280.