UNIVERSITÄT BASEL

An Algorithm for Avoiding Plateaus in Heuristic Search

Bachelor Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence
ai.cs.unibas.ch

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. Martin Wehrle

Claudio Marxer
claudio.marxer@stud.unibas.ch

June 24, 2013

Acknowledgments

This bachelor thesis in Computer Science is written at the Artificial Intelligence Group, University of Basel.

The topic of this thesis was offered by Prof. Dr. Malte Helmert. I would like to thank him for the suggestion of an interesting and instructive project in the field of artificial intelligence.

During my work I was supported by Dr. Martin Wehrle. I would like express to my gratitude for his support and explanations. Thanks for the great supervision.

Last but not least, I would like to express thanks to everyone who supported me during my studies and while working on this project.

Abstract

In action planning, greedy best-first search (GBFS) is one of the standard techniques if suboptimal plans are accepted. GBFS uses a heuristic function to guide the search towards a goal state. To achieve generality, in domain-independant planning the heuristic function is generated automatically. A well-known problem of GBFS are search *plateaus*, i. e., regions in the search space where all states have equal heuristic values. In such regions, heuristic search can degenerate to uninformed search. Hence, techniques to escape from such plateaus are desired to improve the efficiency of the search. A recent approach to avoid plateaus is based on diverse best-first search (DBFS) proposed by Imai and Kishimoto. However, this approach relies on several parameters. This thesis presents an implementation of DBFS into the Fast Downward planner. Furthermore, this thesis presents a systematic evaluation of DBFS for several parameter settings, leading to a better understanding of the impact of the parameter choices to the search performance.

Table of Contents

1 Introduction

The objective of planning is to allow an agent to automatically achieve a desired situation of the world. More precisely, starting in an initial state of the world, the objective of planning is to automatically select a set of actions such that applying these actions is possible and leads to a state that satisfies a certain goal condition.
For illustration, let us look at a common route planning task in public transportation: Assume that an agent has to find a routing from one station to another. To simplify our world, let us assume that there are just train and bus lines. Further, there are two possible actions: If the traveler is in one of these, he can get off at any following station. Elsewise he can get in any train or bus which crosses the current station. The task for the agent is to find a sequence of actions (get on or off a vehicle) that leads the traveler from a given initial station to a given target station.
Problems like these are often modeled as *search problems*. A search problem contains of a set of states (in our example the vehicles or stations on which the traveler can be arranged), a successor function which maps a set of successor states to each state (in our example determined by the actions: get on and off a vehicle) as well as a start state and a set of goal states.
In practice, a successful approach to tackle these kind of search problems is based on *heuristic search* [1]. Therefore a *heuristic function* is defined, which estimates the distance from a given state to a goal state. The heuristic function is then used to find a sequence of actions that leads from the initial state to a goal state.
A very common heuristic search method is *greedy best-first search (GBFS)*. The idea behind GBFS is very intuitive: The search always goes into the direction of a state with lowest heuristic value. By searching in direction of states with shortest estimated distance to the goal, the algorithm tends not to explore unpromising areas of the state space.
It is clear that the performance of GBFS is highly dependent of the accuracy of the heuristic function. An effect caused by inaccurate estimates of the heuristic function in called *search plateau*. Thereby the heuristic value of all available states for exploration are equal (or higher than the heuristic value of the current state). In such a situation, greedy best-first search can not determine a promising search direction, which forces the algorithm to go into arbitrary a direction that is not more promising than any other. It is often the case that this has to

be done several times until a promising direction is found. Therefore, this fashion can cause many unsuccessful tries in order to escape from a search plateau and consequently causes a degradation of performance. Hence, techniques to escape from plateaus are desirable.
Recently, Imai and Kishimoto propose a novel technique for avoiding plateaus [2]. They come up with a search method called *diverse best-first search* (DBFS). In a nutshell, their approach is to diversify the search direction by randomly choosing an node which might have a heuristic value that is higher than the best heuristic value from time to time and performing local searches up to a certain depth starting from this node. Overall, this algorithm relies on various parameters that can be instantiated in various ways. For example, the frequency of choosing an unpromising node is controlled by such a parameter.
The contribution of this bachelor thesis is an implementation and systematic evaluation of DBFS as proposed by Imai and Kishimoto [2] into the Fast Downward planner [3]. In particular, we investigate different parameter configurations and their impact to the algorithm.
The remainder of this thesis is organized as follows. In Chapter 2 we give a formal introduction of planning as well as heuristic search. In that chapter, we also explain the DBFS algorithm. Chapter 3 presents some details about the implementation of the algorithm as well as some comments on the usage of DBFS in Fast Downward. In Chapter 4 the results of our evaluation are presented and discussed. Finally, we give a brief conclusion on the experiments and their discussion in Chapter 5.

2 Preliminaries

In this chapter, planning problems will be introduced. Then, heuristic search as one popular method to tackle these kind of problems is explained.

2.1 Planning

In the literature, several formalisms of planning problems are in use. The following definitions rely on the SAS+ formalism introduced by Bäcktröm and Nebel [4].

Definition (Planning problem):
A *SAS+ planning problem* is a 4-tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ with:

- $\mathcal{V} = \{v_1, \ldots, v_m\}$ denotes a set of *state variables*.

  For each $v \in V$ there exists a *domain* $\mathcal{D}_v$. With u as *undefined value*, ${\mathcal{D}_\mathcal{V}}^+ = \mathcal{D}_\mathcal{V} \cup \{u\}$ denotes the *extended domain*.

  $\mathcal{S}_\mathcal{V} = \mathcal{D}_{v_1} \times \cdots \times \mathcal{D}_{v_m}$ is called the *total state space* of Π, where each $s \in \mathcal{S}_\mathcal{V}$ is called a *state*. In addition, ${\mathcal{S}_\mathcal{V}}^+ = {\mathcal{D}_{v_1}}^+ \times \cdots \times {\mathcal{D}_{v_m}}^+$ denotes the *partial state space* of Π, where $s^+ \in {\mathcal{S}_\mathcal{V}}^+$ is called a *partial state*. For each $s^+ \in {\mathcal{S}_\mathcal{V}}^+$, $s^+[v]$ denotes the state value of v in a state s (where $s[v] = u$ is possible).

- $\mathcal{O} = \{o_1, \ldots, o_n\}$ denotes a set of *operators* of the form $o = \langle pre, post \rangle$, where *pre* denotes the *precondition*, and *post* denotes the effect of o.

- $s_0 \in \mathcal{S}_\mathcal{V}$ is called the *initial state*.

- $s_\star \in {\mathcal{S}_\mathcal{V}}^+$ is called the partial *goal* state.

An operator o is *applicable* in a state s if the precondition of o is satisfied in s. In this case, applying o in s leads to the successor state $o(s)$, which is determined by setting the values of the effect variables of o accordingly.

Definition (Plan):
Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be a planning problem. A *plan* for Π is a sequence of operators $\bar{\alpha} = \langle o_1, \ldots, o_n \rangle$ with $o_k \in \mathcal{O}$ for $1 \leq k \leq n$ such that $\bar{\alpha}$ is applicable in sequence from s_0 and leads to a state that satisfies $s_\star$. $|\bar{\alpha}|$ is called the *length* of a plan $\bar{\alpha}$.
A plan $\bar{\alpha}$ is called *optimal* if no other plan from s to t has fewer operators (if unit cost for each operator are assumed). Otherwise $\bar{\alpha}$ is called a suboptimal plan.

2.2 Heuristic Search

In practice, a planning task as described in the previous section, is usually tackled as a *tree-search problem*. Each node of the corresponding search tree is mapped to a state of the planning task. An edge from a node to a child node denotes an action that can be applied to the state identified with the parent node in order to reach the one identified with the child node. The root node is mapped to the initial state of the planning problem.

A trivial search strategy is to go through the whole search tree systematically. That is how breadth-first search or depth-first search try to find a node that corresponds to a goal state or proof that there is no such node in the tree. In contrast to these so-called uninformed search strategies, there exist *informed search strategies*, which try to use problem-specific knowledge beyond the definition of the problem itself. [5, p. 81]

An instance of informed search is *best-first search*. Algorithm 1 shows pseudocode thereof. Two data structures are used: OpenList (open list) which is a priority queue ordered by a function $f()$ as well as ClosedList which is a set of nodes. At the beginning, ClosedList is empty and OpenList contains only the root node.
Then the actual search starts. The following is done, until a goal node is found or OpenList becomes empty: The minimum node in OpenList is popped from the min-heap and stored in n until the fetched node is not in ClosedList. If n is a goal node the search was successful and the algorithm stops and returns a plan. Otherwise n is added to ClosedList. All child nodes (successors) of n are determined (we will call that expansion) and inserted into OpenList. If the algorithm finds a goal node, a set of actions determined by the path to the root node is returned. Otherwise the open list will become empty at some point which means, that no solution exists.

Now we will have a closer look at lines 1 and 5 of algorithm 1. The *evaluation function* $f()$ determines which node is popped from OpenList and hence, in which direction the search algorithm explores. It is obvious that the choice of the evaluation function is critical for the search effort. If $f()$ leads the algorithm onto a path to a goal node its faster as if unpromising areas of the state space were explored before.

Most best-first search algorithms include as a component of $f(n)$ a function $h(n)$, which is called *heuristic function* [5, p. 92]:

Definition: Let $\mathcal{S}$ be a set of state variables. A *heuristic function* over $\mathcal{S}$ is a function of the following form:

$$h : \mathcal{S} \to \mathbb{R}_0^+$$

Algorithm 1 Best-First Search

```
1: OpenList = new priority queue ordered by f()
2: ClosedList = {}
3: insert the root node into OpenList;
4: while OpenList is not empty do
5:   n := fetch minimum node from OpenList;
6:   if n ∉ ClosedList then
7:     if n is a goal then
8:       return plan to n from the root;
9:     end if
10:    save n in the ClosedList;
11:    expand n;
12:    save successors of n in OpenList;
13:  end if
14: end while
15: return no solution;
```

Russell and Norvig state that "Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm." [5, p. 92] In our setting of planning with heuristic search, "additional knowledge" means, that $h(n)$ gives an estimate of the distance from n to a goal state $s_\star$. The heuristic estimates are computed automatically for a given planning problem.

A very common specialization of best-first search is *greedy best-first search (GBFS)*. GBFS simply uses a heuristic function as evaluation function: $f(n) = h(n)$. Hence, the additional knowledge given from the heuristic function is used to expand the node from OpenList that is expected to lie on the closest path to a goal node.

3

Diverse Best-First Search

The approach of diverse best-first search (DBFS) to escape from search plateaus is to diversify the search direction. From time to time, DBFS selects another node as if greedy best-first search would chose and performs a local search rooted at this node up to a certain depth. Algorithm 2 and 3 show the pseudocode of diverse best-first search as provided by Imai and Kishimoto [2] with some addings.

Algorithm 2 Diverse Best-First Search (Parameter: D)

```
1: insert the root node into OpenList;
2: while OpenList is not empty do
3:    n := fetch a node from OpenList;
4:    LocOL := {n};
5:    /* Local search: Perform GBFS rooted at n */
6:    for i := 1 to h(n) * D do
7:       select node m with 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 successors of m in LocOL;
14:   end for
15:   OpenList := LocOL ∪ OpenList;
16: end while
17: return no solution;
```

The actual search is performed in Algorithm 2. The framework is similar to best-first search. When the search process starts, the data structure OpenList contains solely the root node of the search tree. Then a loop is executed until OpenList becomes empty (that means that there is no solution) or DBFS finds a solution. In that loop, best-first search would fetch a node n, expand it and save the successors to the open list. However, diverse best-first search performs a local search rooted at n. For each local search, a data structure called LocOL (set of nodes) is needed. At the begin of each local search, LocOL contains just the node n. The local search is then peformed by executing $h(n)$ steps (we will call that the depth of the local search) or terminate if a solution is found. In each step of the local

search, the node m with the smallest $h(m)$ is selected and (if it is no goal node) expanded. The successors of m are then inserted into LocOL. If the local search does not find a goal node, thereafter all nodes from LocOL are moved to OpenList. Additionally, we introduce a parameter $D \in \{1, 2, 3, \dots\}$ which is not incorporated in the algorithm of DBFS as proposed by Imai and Kishimoto [2]. The parameter D is a scaling factor for the depth of the local search in Algorithm 2.

Thus, DBDS uses two data structures: The LocOL (local open list) and OpenList (global open list). Fetching a node from the local open list works as fetching a node from the open list in greedy best-first search. It is simply taken the node n with the lowest heuristic value $h(n)$. A node that is fetched from the global open list is calculated by Algorithm 3.

Algorithm 3 Fetching one node (Parameter: P, T)

```
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, ..., h_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;
```

Algorithm 3 deals with an h and g value corresponding to each node. The h-value is equal to the heuristic value of the node while g denotes the sum of the costs of the edges to the root node. The algorithm calculates for each occurrent combination of h and g a probability that a node with these values could be fetched. According to this probability distribution a combination of h and g is selected randomly and a node with these h and g values is returned. If there are multiple possibilities, one of them is selected randomly. The calculation of the random distribution is dependent on the parameters $P \in [0, 1]$ and $T \in [0, 1]$.
With probability of P the probabilities of all nodes with a g-value (distance to root node) greater than a randomly chosen threshold is set to zero. This forces the algorithm to fetch a node as starting point for the local search which does not exceed a certain distance to the root node. The probability of a pair of h and g (except the ones excluded by P) to be chosen, is determined by parameter T and by h. The smaller the difference between h and the lowest h value, the higher is the probability. Hence, nodes with lower heuristic values

are preferred. The grad of preference is controlled by T. The lower T, the higher is the probability of a node with a low h value to be chosen.
With this framework it is possible to fetch an unpromising node from the global open list in a "controlled manner".

4 Implementation

We implemented diverse best-first search (DBFS) as described in the previous chapter into the Fast Downward planner by Helmert [3]. This chapter gives an overview of Fast Downward (FD) and the implementation of DBFS.

4.1 Fast Downward

Fast Downward is a heuristic search planner implemented in C++ and Python. It can deal with general planning problems as defined in Section 2.1. The input is encoded in the planning domain definition language (PDDL2.2) [6] as well as some fragments of newer versions. Fast Downward implements different search algorithms and heuristic functions.

4.2 Implementation

Diverse best-first search is implemented as a new search engine `dbfs_search` inherited from `eager_greedy`. The last-mentioned search engine implements greedy best-first search and modifications thereof.
For the global open list we implemented a new open list called `dbfs_open_list`. The code thereof relies on the `standard_scalar_open_list`, which is also used as the local open list. For all other tasks Fast Downward is used as a framework. DBFS can be combined with all available heuristic functions.

4.3 Usage

As the implementation of DBFS is done into the Fast Downward planner, the call is the same as with any other search algorithm[1].

The Fast Downward planner runs in three steps. [3, p. 202-203] The first two steps, translating and preprocessing, are not explained here. The actual search process starts in the

[1] http://www.fast-downward.org/PlannerUsage

third step, where the search method is performed.

After processing the first two steps, a file named `output` is generated. Then, DBFS is called like this:

```
./downward --search "dbfs(ff())" < output
```

In this example, DBFS uses the FF heuristic. More information about the available heuristics can be found on the web page of the Fast Downward project[2].

If no additional parameters are provided, DBFS uses a standard configuration: $P = 0.1$, $T = 0.5$ and $D = 1$. In the following example DBFS is called with specified parameters:

```
./downward --search "dbfs(ff(),p=0.2,t=0.4,d=2)" < output
```

As defined in all previous chapter, `p` and `t` accept `float`-values in $[0, 1]$. `d` as a scaling factor of the depth of the local search accepts positive `int`-values.

To turn off the local search, the planner can be called like:

```
./downward --search "dbfs(ff(),d=0)" < output
```

In that case, DBFS behaves like greedy best-first search just with fetching the next node by using algorithm 2.3 instead of taking that one with the lowest heuristic value.

Because the DBFS algorithm works with random values, different runs with the same configuration usually produce different outputs. To get reproducible results, Fast Downward can be random seeded:

```
./downward --search "dbfs(ff())" --random-seed 123622312 < output
```

[2] http://www.fast-downward.org/HeuristicSpecification

5 Evaluation

This chapter contains the results of our evaluations and a discussion of these.

5.1 Experimental Setup

We did experiments with all benchmarks of the International Planning Competition until the IPC11 which contains 2252 problems in 58 domains. Each domain include at least 5 and at most 80 problems.

All tests ran on Intel Xeon E5-2660 CPUs (2.2 GHz) with a memory and time limit of 2 GB and 30 minutes. Both, the memory and time limit exceeded in some runs.

Diverse best-first search incorporated random numbers. Thus, different outputs can be produced for the same parameters on the same instance. That is why we performed several passages. If averages thereof are computed, the result is rounded to the nearest integer. For comparison the non-rounded values are used.
To get reproducible results, we ran all experiments with random seeds. A random seed determines the calculation of the pseudo random numbers used by DBFS and therefore also the search result. All random seeds we used for our experiments are listed in Appendix.

We used the experiment framework lab[1] to conduct and analyze the experiments.

[1] http://lab.readthedocs.org

5.2 Results

In this section, we present the results of our evaluations.

5.2.1 Variation of a Single Parameter

We started the evaluation with the standard configuration provided by Imai and Kishimoto [2]. They set $P = 0.1$ and $T = 0.5$. Since the parameter D as a scaling factor of the local search is not introduced in their algorithm, we set $D = 1$ as standard.
First, we varied a single parameter, while the others were fixed on the standard configuration. P took all values in $\{0.1, 0.2, 0.3\}$, T all values in $\{0.4, 0.5, 0.6\}$ while D varied in $\{0, 1, 2, 3, 5\}$ where $D = 0$ means that there is no local search performed. If the best results performed with a parameter on a bound of the range, we extended the range with some additional values.
We ran each test with five different random seeds. All experiments were performed with three different heuristic functions. We used the FF heuristic [7], the context-enhanced additive (CEA) heuristic [8] and the causal graph (CG) heuristic [9]. All of them are already implemented in Fast Downward.
Table 5.1 shows the coverage of FF, Table 5.2 the coverage of CEA and Table 5.3 the coverage of CG. The configurations marked with $\star$ have the best average of solved instances over all seeds in the particular range.

Configuration		**seed1**	**seed2**	**seed3**	**seed4**	**seed5**	**Min.**	**Max.**	**Avg**
GBFS		1576					1576	1576	1576
DBFS (P=0.1)		1769	1777	1770	1776	1773	1769	1777	1773
DBFS (P=0.2)		1773	1779	1779	1777	1776	1773	1779	1777
DBFS (P=0.3)	$\star$	1773	1780	1780	1778	1775	1773	1780	**1777**
DBFS (P=0.4)		1763	1779	1766	1754	1765	1754	1779	1765
DBFS (P=0.5)		1768	1764	1762	1760	1772	1760	1772	1765
DBFS (T=0.4)		1771	1759	1762	1769	1772	1759	1772	1767
DBFS (T=0.5)		1768	1777	1769	1776	1773	1768	1777	1773
DBFS (T=0.6)	$\star$	1786	1780	1772	1768	1774	1768	1786	**1776**
DBFS (T=0.7)		1780	1769	1771	1767	1773	1767	1780	1772
DBFS (T=0.8)		1750	1747	1752	1750	1746	1746	1752	1749
DBFS (D=1)		1769	1776	1769	1776	1773	1769	1776	1773
DBFS (D=2)		1798	1813	1800	1805	1784	1784	1813	1800
DBFS (D=3)		1801	1801	1793	1789	1792	1789	1801	1795
DBFS (D=4)		1799	1802	1804	1805	1808	1799	1808	1804
DBFS (D=5)		1800	1803	1812	1801	1801	1800	1812	1803
DBFS (D=6)	$\star$	1810	1806	1806	1807	1805	1805	1810	**1807**
DBFS (D=7)		1799	1806	1810	1811	1799	1799	1811	1805
DBFS (D=0)		1579	1575	1587	1571	1583	1571	1587	1579

Table 5.1: Coverage of DBFS and FF heuristic with different configurations and random seeds.

Configuration		seed1	seed2	seed3	seed4	seed5	Min.	Max.	Avg
GBFS		1595					1595	1595	1595
DBFS (P=0.1)	⋆	1812	1799	1793	1806	1790	1790	1812	**1800**
DBFS (P=0.2)		1786	1787	1793	1785	1784	1784	1793	1787
DBFS (P=0.3)		1786	1772	1768	1769	1771	1768	1786	1773
DBFS (T=0.4)		1789	1790	1804	1791	1786	1786	1804	1792
DBFS (T=0.5)	⋆	1812	1800	1793	1809	1790	1790	1812	**1801**
DBFS (T=0.6)		1803	1797	1799	1802	1795	1795	1803	1799
DBFS (D=1)	⋆	1811	1800	1794	1806	1791	1791	1811	**1800**
DBFS (D=2)		1798	1787	1786	1790	1788	1786	1798	1790
DBFS (D=3)		1780	1786	1790	1787	1783	1780	1790	1785
DBFS (D=5)		1765	1767	1766	1771	1767	1765	1771	1767
DBFS (D=0)		1563	1560	1565	1560	1549	1549	1563	1559

Table 5.2: Coverage of DBFS and CEA heuristic with different configurations and random seeds.

Configuration		seed1	seed2	seed3	seed4	seed5	Min.	Max.	Avg
GBFS		1592					1592	1592	1592
DBFS (P=0.1)		1667	1654	1664	1674	1671	1654	1674	1666
DBFS (P=0.2)	⋆	1665	1678	1663	1661	1669	1661	1678	**1667**
DBFS (P=0.3)		1662	1660	1658	1668	1665	1658	1668	1663
DBFS (T=0.4)		1652	1671	1678	1659	1668	1652	1678	1666
DBFS (T=0.5)	⋆	1667	1655	1664	1674	1672	1655	1674	**1666**
DBFS (T=0.6)		1661	1661	1668	1657	1660	1657	1668	1661
DBFS (D=1)		1668	1654	1664	1674	1672	1654	1674	1666
DBFS (D=2)		1698	1688	1683	1678	1680	1678	1698	1685
DBFS (D=3)		1686	1697	1677	1693	1691	1677	1697	1689
DBFS (D=4)		1690	1694	1692	1695	1689	1689	1695	1692
DBFS (D=5)		1703	1688	1700	1707	1705	1688	1707	1701
DBFS (D=6)		1692	1701	1696	1710	1694	1692	1710	1699
DBFS (D=7)	⋆	1705	1707	1700	1708	1696	1696	1708	**1703**
DBFS (D=8)		1702	1696	1695	1698	1704	1695	1704	1699
DBFS (D=9)		1703	1710	1696	1698	1696	1696	1710	1701
DBFS (D=0)		1475	1484	1469	1469	1470	1469	1484	1473

Table 5.3: Coverage of DBFS and CG heuristic with different configurations and random seeds.

5.2.2 Combination of Best Results

After we have evaluated the best configuration by varying only one parameter while the others were fixed, we combined the P, T and D on which the experiments performed best.

Configuration	seed1	seed2	seed3	seed4	seed5	Min.	Max.	Avg
GBFS	1576					1576	1576	1576
DBFS (P=0.3)	1773	1780	1780	1778	1775	1773	1780	1777
DBFS (T=0.6)	1786	1780	1772	1768	1774	1768	1786	1776
DBFS (D=6)	1810	1806	1806	1807	1805	1805	1810	**1807**
DBFS (Combination)	1798	1793	1796	1804	1800	1793	1804	1798

Table 5.4: Coverage of DBFS and FF heuristic with a combination of the best parameters.

Configuration	seed1	seed2	seed3	seed4	seed5	Min.	Max.	Avg
GBFS	1595					1595	1595	1595
DBFS (P=0.1)	1812	1799	1793	1806	1790	1790	1812	1800
DBFS (T=0.5)	1812	1800	1793	1809	1790	1790	1812	1801
DBFS (D=1)	1811	1800	1794	1806	1791	1791	1811	1800
DBFS (Combination)	1814	1801	1793	1808	1792	1792	1814	**1802**

Table 5.5: Coverage of DBFS and CEA heuristic with a combination of the best parameters.

Configuration	seed1	seed2	seed3	seed4	seed5	Min.	Max.	Avg
GBFS	1592					1592	1592	1592
DBFS (P=0.2)	1665	1678	1663	1661	1669	1661	1678	1667
DBFS (T=0.5)	1667	1655	1664	1674	1672	1655	1674	1666
DBFS (D=7)	1705	1707	1700	1708	1696	1696	1708	**1703**
DBFS (Combination)	1696	1700	1701	1701	1692	1692	1702	1698

Table 5.6: Coverage of DBFS and CG heuristic with a combination of the best parameters.

5.2.3 Coverage on Domain Level

Table 5.7 lists the parameter configurations we observed in all previous experiments for each heuristic function which solved most instances on average.

Heuristic	*P*	*T*	*D*	Avg
FF	0.1	0.5	6	1807
CEA	0.1	0.5	1	1802
CG	0.1	0.5	7	1703

Table 5.7: Best observed parameter configurations for each heuristic function.

Table 5.8 shows the coverage on domain level of the best parameter configuration for DBFS with the FF heuristics. Because experiments with the same configuration and different seeds gives slightly different results, we list the values of the run with the median coverage.

Domain	DBFS with FF	GBFS with FF
airport **(50)**	**36**	34
assembly **(30)**	**30**	20
barman-sat11-strips **(20)**	**16**	2
blocks **(35)**	**35**	**35**
depot **(22)**	**17**	15
driverlog **(20)**	**20**	18
elevators-sat08-strips **(30)**	**13**	11
elevators-sat11-strips **(20)**	**0**	**0**
floortile-sat11-strips **(20)**	**8**	7
freecell **(80)**	**80**	79
grid **(5)**	**4**	**4**
gripper **(20)**	**20**	**20**
logistics00 **(28)**	**28**	**28**
logistics98 **(35)**	26	**30**
miconic **(150)**	**150**	**150**
miconic-fulladl **(150)**	**139**	136
miconic-simpleadl **(150)**	**150**	**150**
movie **(30)**	**30**	**30**
mprime **(35)**	**34**	31
mystery **(30)**	**19**	17
nomystery-sat11-strips **(20)**	**15**	10
openstacks **(30)**	**30**	**30**
openstacks-sat08-adl **(30)**	5	**6**
openstacks-sat08-strips **(30)**	5	**6**
openstacks-sat11-strips **(20)**	**0**	**0**
openstacks-strips **(30)**	**30**	**30**
optical-telegraphs **(48)**	**14**	4
parcprinter-08-strips **(30)**	**22**	**22**
parcprinter-sat11-strips **(20)**	**5**	**5**
parking-sat11-strips **(20)**	**20**	**20**
pathways **(30)**	**25**	10
pathways-noneg **(30)**	**26**	11
pegsol-08-strips **(30)**	**30**	**30**
pegsol-sat11-strips **(20)**	**20**	**20**
philosophers **(48)**	**48**	**48**
pipesworld-notankage **(50)**	**42**	33
pipesworld-tankage **(50)**	**29**	21
psr-large **(50)**	**18**	13
psr-middle **(50)**	**45**	38
psr-small **(50)**	**50**	**50**
rovers **(40)**	**32**	23
satellite **(36)**	**28**	27
scanalyzer-08-strips **(30)**	**30**	28
scanalyzer-sat11-strips **(20)**	**20**	18

schedule **(150)**	**121**	37
sokoban-sat08-strips **(30)**	**28**	**28**
sokoban-sat11-strips **(20)**	**18**	**18**
storage **(30)**	**24**	18
tidybot-sat11-strips **(20)**	**18**	15
tpp **(30)**	**25**	23
transport-sat08-strips **(30)**	**14**	11
transport-sat11-strips **(20)**	**0**	**0**
trucks **(30)**	**21**	17
trucks-strips **(30)**	**22**	17
visitall-sat11-strips **(20)**	**3**	**3**
woodworking-sat08-strips **(30)**	**30**	27
woodworking-sat11-strips **(20)**	**18**	12
zenotravel **(20)**	**20**	20
Sum 2252	**1806**	**1576**

Table 5.8: Coverage on domain level of DBFS and the FF heuristic with the best observed configuration compared to GBFS with FF.

5.2.4 Search Time

We compared the search time of GBFS and DBFS with the parameter configurations listed in Table 5.7. Figure 5.1, 5.2 and 5.3 illustrate the results for FF, CEA and CG. The search time measures only the time it takes to perform the actual search. Other tasks (e.g. preprocessing) are not included.

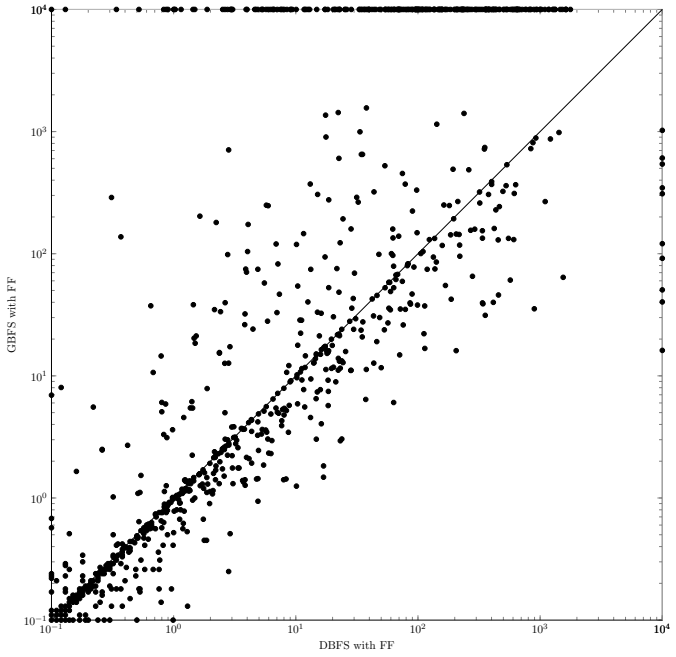

Figure 5.1: Comparison of the search time of DBFS and GBFS with the FF heuristic.

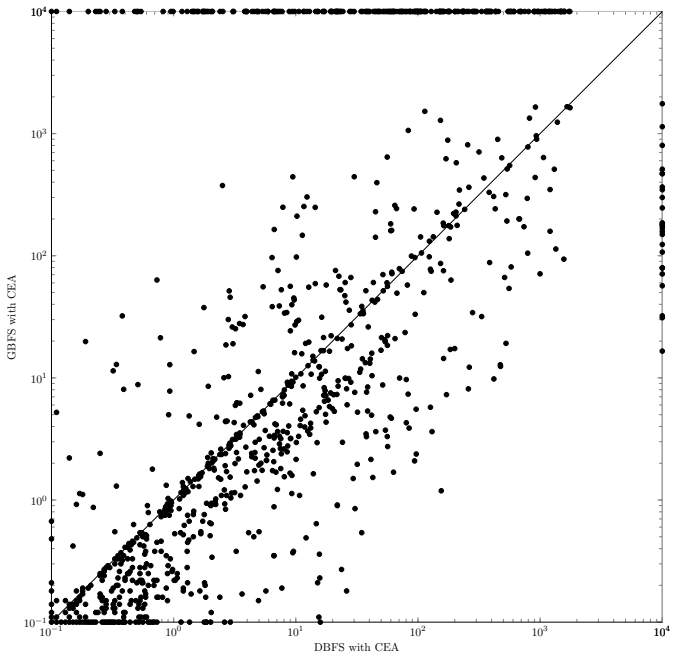

Figure 5.2: Comparison of the search time of DBFS and GBFS with the CEA heuristic.

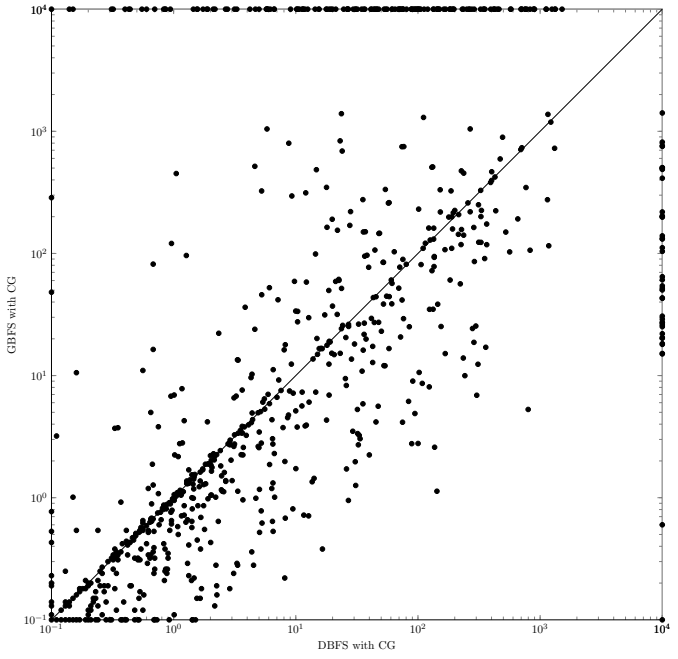

Figure 5.3: Comparison of the search time of DBFS and GBFS with the CG heuristic.

5.2.5 Plan Cost

Figure 5.4, 5.5 and 5.6 compare the cost of the plans that GBFS and DBFS found with the parameter configurations as listed in Table 5.7.

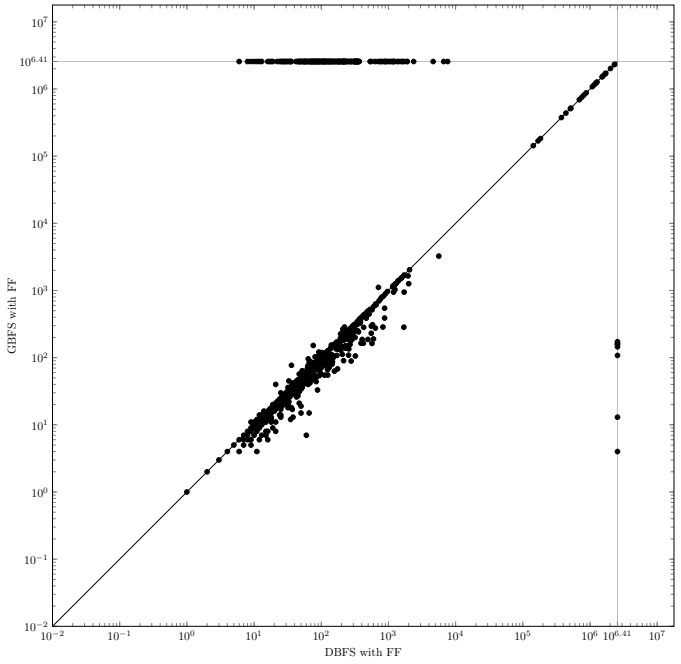

Figure 5.4: Comparison of the plan cost of DBFS and GBFS with the FF heuristic.

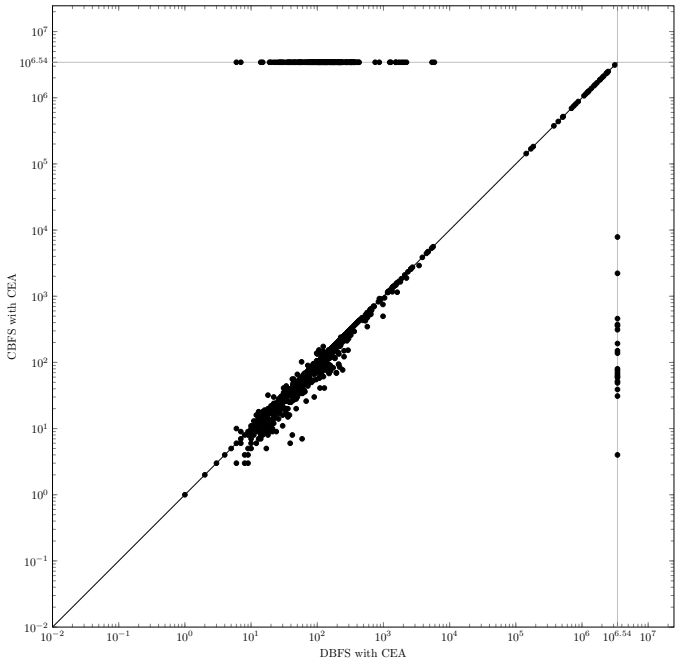

Figure 5.5: Comparison of the plan cost of DBFS and GBFS with the CEA heuristic.

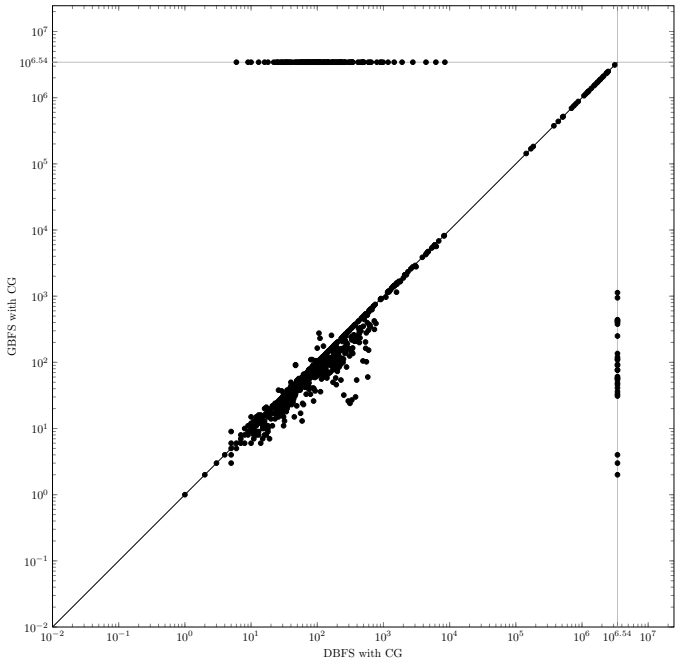

Figure 5.6: Comparison of the plan cost of DBFS and GBFS with the CG heuristic.

5.3 Discussion

In the first experiments we varied a single parameter, while the others were fixed. Diverse best-first search solves with each parameter configuration (except when the local search was turned off) more instances than greedy best-first search. While GBFS with the FF and the CEA heuristic solves fewer that 1600 of 2252 problems, no configuration of DBFS with these heuristic functions results in a coverage under 1749 instances. That is an improvement of at least 150 instances. On average, the performance improvement with CG is lower. However, even with the poorest configuration, DBFS solves 69 instances more than GBFS.
For all heuristic functions the impact of the parameters P and T seem not to be very strong. Running DBFS in combination with the FF and the CG heuristic, the parameter D has a substantial impact: Using the FF heuristic, the coverage improves by 34 instances while using the CG heuristic the coverage increases by 37 instances. With the CEA heuristic the best coverage performs without any scaling of the depth of the local search.
The experiments show that the coverage can not be improved, if the local search makes only one step ($D = 0$). With the FF heuristic, DBFS solves 3 instances more than GBFS. With the CEA and CG heuristics, DBFS covers even less problems.
Imai and Kishimoto evaluated $P = 0.1$ and $T = 0.5$ as the parameter configuration with the best coverage. Our results mainly confirm that: For all heuristic functions, these parameters solve most instances or lie very close to the coverage of the best configuration.
Imai and Kishimoto state that DBFS "solves either an equal or larger number of instances in all the domains". We evaluated the coverage on domain level for the best configuration we observed with the FF heuristic. In contrary to the evaluations of Imai and Kishimoto, GBFS solves more instances than DBFS in the domain Logistics 1998. In the domains openstacks-sat08-strips and openstacks-sat11-strips which were not evaluated by

Imai and Kishimoto, GBFS performs also better than DBFS. In all other 55 domains GBFS do not cover more instances than DBFS.

The scatter plots who compare the search time of DBFS and GBFS look for all heuristic functions very similar. Most points lie close to the diagonal which means that the search time of all instances solved by both methods does not vary greatly. By trend, the search time with DBFS is longer.

The scatter plots who compare the plan cost of DBFS and GBFS look for all heuristic functions very similar as well. The points are clustered around the diagonal without much outliers. The cluster tend to lie more on the side of DBFS which means that plans of GBFS have fewer cost in general. However, the influence on the plan cost are not high.

6 Conclusion

In this thesis, we have addressed diverse best-first search (DBFS), a recently proposed algorithm for escaping from plateaus in the context of planning as heuristic search. We have implemented DBFS in the Fast Downward planner, and provided an empirical evaluation of DBFS on benchmarks from the international planning competitions. In particular, we have investigated different parameter configurations of DBFS and how these parameter configurations influence the search performance.

Our experiments show that DBFS outperforms greedy best first search in most of the planning domains for several well-known planning heuristics. The parameter choices appear to influence the search performance only marginally. Moreover, our experiments show that the local searches performed by GBFS seem to be more important for the performance than the randomness component. Overall, DBFS appears to be an effective and efficient algorithm for satisficing planning.

Bibliography

[1] Bonet, B. and Geffner, H. Planning as Heuristic Search. *Artificial Intelligence*, 129:5–33 (2001).

[2] Imai, T. and Kishimoto, A. A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI 2011)*, pages 985–991 (2011).

[3] Helmert, M. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26:191–246 (2006).

[4] Bäckström, C. and Nebel, B. Complexity Results for SAS+ Planning. *Computational Intelligence*, 11:625–655 (1993).

[5] Russell, S. J. and Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition (2009).

[6] Edelkamp, S. and Hoffmann, J. PDDL2.2: The language for the Classical Part of the 4th International Planning Competition. IPC-4 Booklet (2004).

[7] Hoffmann, J. and Nebel, B. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302 (2001).

[8] Helmert, M. and Geffner, H. Unifying the causal graph and additive heuristics. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008)*, pages 140–147 (2008).

[9] Helmert, M. A Planning Heuristic Based on Causal Graph Analysis. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 161–170 (2004).

Appendix

Random Seeds

seed1	123622312
seed2	943721659
seed3	578590589
seed4	899239028
seed5	462307820

Declaration of Authorship

I hereby declare that this thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the bibliography and specified in the text.
This thesis is not substantially the same as any that I have submitted or will be submitting for a degree or diploma or other qualification at this or any other University.

Basel, 24th June 2013

Claudio Marxer