博士論文

Diversification Mechanisms for Best-First Search
(最良優先探索のための探索非局在化手法)

浅井 政太郎

Masataro Asai

Abstract

Despite the recent advances in domain-independent planning algorithms, there is still a large gap between the theory and practice of search algorithms for planning. In cost-optimal search, despite the major advances in lower bound functions (heuristic functions), the study of the base algorithm itself is rarely attempted recently. State-of-the-art satisficing search algorithms use complex combinations of various, ad-hoc search enhancements, making the resulting algorithm difficult to analyze. The relation between search algorithms guaranteed to find the optimal solution and satisficing search technique has also not been investigated in depth. This dissertation proposes a unified framework for understanding these algorithms, based upon which several new algorithmic enhancements are proposed to improve the state of the art.

We first analyze and improve the tiebreaking behavior of A*, the standard algorithm for cost-optimal search. We develop a new framework for viewing cost-optimal search as a series of satisficing search episodes, and show that this new perspective can be effectively exploited by new tie-breaking strategies which significantly improve upon standard tie-breaking strategies.

Having established effective satisficing search as a key component of cost-optimal search, we then focus on methods for improving satisficing search. We

unify previous approaches for diversitying satisfying search as instances of orthogonal, inter- and intra-plateau diversification. We show that this new perspective leads to effective, new combinations of diversification strategies which improve upon the state-of-the-art diversifciation strategies. We also propose Invasion Percolation, a new fractal-inspired diversification method which complements previous diversification approaches.

Preface

This thesis is based on author's past conference and journal publications. Chapter 3-Chapter 5 are published in (Asai & Fukunaga, 2016) and (Asai & Fukunaga, 2017b). Chapter 6-Chapter 7 are published in (Asai & Fukunaga, 2017a).

Contents

Abstract **iii**

Preface **v**

1 Introduction **1**

2 Background **4**

2.1 Classical Planning 4

2.2 Basic Search Algorithms and Notation 7

3 Tie-Breaking Strategies and Plateau Structure **11**

3.1 Tie-Breaking Strategies for A^* 14

3.2 Analysis of Standard Strategies 17

3.2.1 Is h-Based Tie-Breaking Necessary? 18

3.2.2 Do Default Strategies Make a Difference? 20

3.2.3 Plateaus and Tie-Breaking 20

3.3 Domains with 0-Cost Actions 22

3.3.1 Difference in Problem Characteristics between IPC and Zerocost Domains 27

4 Depth Diversification **31**

4.1 Depth-Based Tie-Breaking for A* 34

4.1.1 Tie-Breaking within Depth Buckets 37

4.1.2 Theoretical Characteristics of the Depth Distribution . . . 39

4.2 Evaluating Depth-Based Tie-Breaking 42

4.2.1 Search Behavior Within a Plateau 48

4.2.2 The Effect of Domain Mangling 51

5 Optimal Search as Satisficing Search **55**

5.1 Depth Diversification as Satisficing Search 58

5.2 Completeness of A^* on an Infinite Graph 60

5.3 Tie-Breaking Using Distance-to-Go Estimates 63

5.4 Embedding Distance-to-Go in Admissible Search 65

5.4.1 Distance-to-Go Estimates with Default Tie-Breaking . . . 66

5.4.2 Distance-to-Go Estimates with Depth Diversification . . . 66

5.5 Evaluation 67

5.5.1 Evaluation on Zerocost Domains 67

5.5.2 Evaluation on Standard IPC Domains 69

5.5.3 Summary of the Evaluation 70

5.5.4 Simple Dynamic Configuration for Overall Performance . 70

6 Intra-vs-Inter Plateau Diversification **74**

6.1 Background 77

6.1.1 Exploration Mechanisms 77

6.1.2 Tiebreaking 78

6.2 Intra- and Inter-plateau Diversification 79

6.2.1 Type-Based Diversification 81
6.3 Empirical Comparison 82

7 Invasion Percolation **87**
7.1 Invasion Percolation 89
7.2 Invasion Percolation for Search Diversification 91
7.3 Search Behavior of IP-diversification 93
7.4 Intra- and Inter-Plateau Diversification on a State-of-the-Art Planner 95
7.5 Evaluation of IP-Diversification 97

8 Related Work **101**

9 Conclusion and Future Work **105**

Appendix: Detailed Data **109**
9.1 Detailed Data for Table 3.2.1 110
9.2 Detailed Data for Table 4.2.1 112
9.3 Additional Figures for Figure 4.2.1 116
9.4 Additional Figures for Figure 4.2.3 117
9.5 Additional Figures for Figure 4.2.4 119
9.6 Detailed Data for Table 5.5.2 120

Bibliography **124**

Chapter 1

Introduction

Over the years, heuristic search based mehods for automated planning has achieved significant success and shown its ability to scale to larger and larger problems. Much of the success can be attributed to the development of increasingly sophisticated heuristic functions, while relatively less attention was paid to the base search algorithms.

In fact, despite the number of papers which try to push the state of the art in optimal planning by improving admissible heuristic functions and developing their theory, much less attention has been paid to the common underlying algorithm, A*, until (Asai & Fukunaga, 2016). Similarly, while there is a large body of work on satisficing planning algorithms, many algorithms tend to be ad-hoc and lack theoretical foundation other than completeness. For example, the state-of-the-art LAMA planner (Richter & Westphal, 2010) incorporates five search algorithm-related improvements at once and the reason for its success on benchmark domains is not yet fully understood.

The contribution of this dissertation is a new framework for understanding

search algorithm behavior in terms of their behavior on the critical, frontier region of the search space, and proposals of new algorithms based on the new understanding. This dissertation proceeds as follows.

After the introduction and the preliminary background, we first analyze and discuss the search space topology of various domain-independent planning problems with regard to f, the admissible lower-bound for the solution cost (Chapter 3). We show that, contrary to the conventional wisdom, these combinatorial problems contain huge *final plateaus*, a set of nodes which have the same f value as the optimal solution cost f^*. We next investigate the behavior of existing tiebreaking strategy for A^* algorithm and identify an important class of problems called *Zerocost domains*, which are characterized by a huge number of zero-cost edges and renders existing tiebreaking strategies useless.

In the next chapter (Chapter 4), we propose a notion of *depth* in a plateau that explains the behavior of existing tiebreaking strategies in Zerocost domains. We then propose a new strategy called *depth diversification* which significantly outperforms the existing strategies in several zero-cost domains. We analyze the behavior of depth diversification under some assumptions and verify that the empirical behavior is consistent with expectations. Thus, we show that, in domain-independent planning, there is still plenty room for improvements in the base search algorithms that can impact the search performance.

In Chapter 5, based upon the findings in the previous section, we proceed to show that optimal search can be reduced to satisficing search. We reformulate the traditional understanding of optimal [1] best-first search algorithms (such as A*) by dividing the search space into *plateaus* of increasing f-value, then characterizing

[1] An optimal search algorithm is guaranteed to return the optimal-cost solution.

A* as a sequence of satisficing searches on each plateau in the increasing order of f-value.

Chapter 5 effectively shows that the performance of optimal search algorithms can be improved by improving the underlying satisficing search which is being performed in every plateau layer. Thus, the rest of the dissertation focuses on satisficing search. To obtain a deeper understanding of satisficing search algorithm, in Chapter 6, we investigate two notions in search algorithms, tiebreaking and exploration, and reformulate them as orthogonal approaches to address the errors between a heuristic function h and the true cost to goal h^*. We empirically verify this hypothesis by comparing the search performance between algorithms where the same diversification mechanism is applied to either tiebreaking or h-value selection, or both.

Since the diversification mechanism in both tiebreaking and exploration are based on *knowledge-free*, blind search algorithms, we further conclude that satisficing search algorithms can be ultimately improved by developing the more sophisticated blind search algorithms. In Chapter 7, we propose a new diversification mechanism called *Invasion Percolation*, which is based on a fractal structure resulting from the minimum spanning tree on a search graph.

The dissertation concludes by discussing the relationship with related work and discussing directions for future work.

Chapter 2

Background

2.1 Classical Planning

Classical Planning has achieved significant advances in recent years due to the success of heuristic search based methods. The input problem to a Classical Planning solver (a *planner*) is a 5-tuple $\Pi = \langle P, O, I, G, A \rangle$ where P defines a set of first-order predicates, O is a set of symbols called *objects*, I is the initial state, G is a set of goal conditions, and A is a set of actions which defines the state transitions in the search space. A state is an assignment of boolean values to the set of propositional variables, while a condition is a partial assignment that assigns values only to a subset of propositions. Each proposition is an instantiation of a predicate with objects. Lifted action schema $a \in A$ is a 5-tuple $\langle \textit{params}, \textit{pre}, e^+, e^-, c \rangle$ where each element denotes the set of parameters, preconditions, add-effects, delete-effects and the cost, respectively. Parameter substitution using objects in O instantiates *ground actions*. When c is not specified, it is usually assumed $c = 1$. These inputs are described in the Planning Domain Description Language (PDDL)

(McDermott, 2000) and its extensions.

Figure 2.1.1 shows one possible representation of a state in the 3x3 sliding tile puzzle (8-puzzle) domain as a First Order Logic formula, and the representation of the same state using PDDL.

$$
\begin{aligned}
&Empty(x_0, y_0) \\
\wedge &At(x_1, y_0, panel_6) \\
\wedge &Up(y_0, y_1) \\
\wedge &Down(y_1, y_0) \\
\wedge &Right(x_0, x_1) \\
\wedge &Left(x_1, x_0) \ldots
\end{aligned}
$$

```
(empty x0 y0)
(at    x1 y0 panel6)
(up    y0 y1)
(down  y1 y0)
(right x0 x1)
(left  x1 x0)...
```

	6	8
7	3	2
5	1	4

Figure 2.1.1: One possible state representation of a 3x3 sliding tile puzzle (8-puzzle) as a first order logic formula and its corresponding PDDL notation. It contains predicate symbols empty, up, down, left, right, at as well as object symbols such as x_i, y_i, panel_j for $i \in \{0..2\}$ and $j \in \{1..8\}$.

The task of a planning problem is to find a path from the initial state I to some goal state $s^* \supseteq G$, using the state transition rules in A. A state s can be transformed into a new state t by applying a ground action a when $s \supseteq$ *pre*, and then $t = (s \setminus e^-) \cup e^+$ (McDermott, 2000). This transition can also be viewed as applying a state transition function a to s, which can be written as $t = a(s)$.

State-of-the-Art planners solve this problem as a path finding problem on an implicit graph defined by the state transition rules. They usually employ forward state space heuristic search, such as A^* (for finding the shortest path) or Greedy Best-First Search (for finding a suboptimal path more quickly). Due to a variety of

When $Empty(x, y_{old})$
$\wedge\, at(x, y_{new}, p)$
$\wedge\, up(y_{new}, y_{old})$;
then $\neg Empty(x, y_{old})$
$\wedge\, Empty(x, y_{new})$
$\wedge\, \neg at(x, y_{new}, p)$
$\wedge\, at(x, y_{old}, p)$

```
(:action slide-up ...
 :precondition
 (and (empty ?x ?y-old)
      (at ?x ?y-new ?p)
      (up ?y-new ?y-old))
 :effects
 (and (not (empty ?x ?y-old))
      (empty ?x ?y-new)
      (not (at ?x ?y-new ?p))
      (at ?x ?y-old ?p)))
```

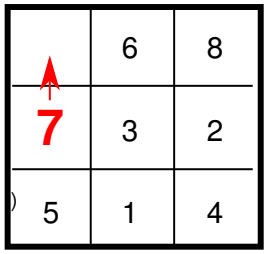

Figure 2.1.2: One possible action representation of sliding up a tile in 3x3 sliding tile puzzle in (left) the first order logic formula and (middle) its corresponding PDDL notation. In addition to Figure 2.1.1, it further contains an action symbol slide-up.

successful domain-independent heuristic functions (Helmert & Domshlak, 2009; Sievers, Ortlieb, & Helmert, 2012; Helmert, Haslum, & Hoffmann, 2007; Bonet, 2013; Hoffmann & Nebel, 2001; Helmert, 2004; Richter, Helmert, & Westphal, 2008), current state-of-the-art planners can scale to larger problems which requires to find a plan consisting of more than 1000 steps (Asai & Fukunaga, 2015).

While evaluating performance of a planner, we sometimes measure its "coverage", i.e. the number of instances solved in a particular resource limitation among a certain set of instances. Coverage is one of the popular metric for measuring the performance of a planner.

2.2 Basic Search Algorithms and Notation

A^* is a standard search algorithm for finding an optimal cost path from an initial state s to some goal state $s^* \in G$ in a search space represented as a graph (Hart, Nilsson, & Raphael, 1968). It expands the nodes in best-first order of $f(n)$ up to f^*, where $f(n)$ is a lower bound of the cost of the shortest path that contains a node n and f^* is the cost of the optimal path. The value of $f(n)$ is a sum of $g(n)$, the known shortest path cost so far from the initial node to n, and $h(n)$, the heuristic lower-bound estimate of the cost from n to some goal s^*. $h(n)$ is *admissible* if it does not overestimate the true cost to goal $h^*(n)$, which is also called a *perfect heuristic*. We omit the argument (n) unless necessary. For domain-independent classical planning, notable state-of-the-art, admissible heuristic functions are LM-cut (Helmert & Domshlak, 2009) and Merge-and-Shrink (Helmert, Haslum, Hoffmann, & Nissim, 2014).

Greedy Best First Search (GBFS) is a greedy search algorithm that is intended to find a satisficing solution to the problem as quickly as possible, without explicitly trying to minimize the path cost. It expands the nodes in the best-first order of $h(n)$, i.e. greedily guided by the lower-bound estimate. Since GBFS does not guarantee the cost optimality (the worst case solution cost is unbounded), GBFS tends to be used together with an *inadmissible* heuristic function that may overestimate the goal. Notable state-of-the-art inadmissible heuristics include FF heuristic (Hoffmann & Nebel, 2001) and Causal Graph heuristic (Helmert, 2004).

Both search algorithms can be described using a uniform notation, which we call a *sorting strategy*. Below, we first present a general *Best First Search* (BFS) algorithm template which includes A^*, Dijkstra's algorithm (1959), Greedy Best-First Search (GBFS). It uses two sets, OPEN and CLOSED, where unexpanded

nodes are stored in OPEN and expanded nodes are stored in CLOSED. Three operations, pop(S), push(n, S) and remove(n, S), are assumed for a node n and a set S. pop(S) operation tries to select a single node from S, push(n, S) stores the node n into S and remove(n, S) removes n from S if n is already stored.

Algorithm 1 Best-First Search Algorithm using OPEN/CLOSED list

Input: n_0, is_goal($\cdot$), successors($\cdot$)

1: Initialize OPEN $= \emptyset$, CLOSED $= \emptyset$, $g(n_0) = 0$, $(\forall n \neq n_0;\ g(n) = \infty)$
2: push(n_0, OPEN)
3: **while** OPEN $\neq \emptyset$ **do**
4: $n =$ pop(OPEN); push(n, CLOSED)
5: **return** n **if** is_goal(n) = true
6: **for each** $m \in$ successors(n) **do**
7: $g_{new} = g(n) +$ cost(n, m)
8: **if** $g_{new} < g(m)$ **then**
9: $g(m) \leftarrow g_{new}$; parent(m) $\leftarrow n$; push(m, OPEN); remove(m, CLOSED)

OPEN is sorted according to a *sorting strategy* and the node selected by pop(S) always returns the best node according to the strategy. Each sorting strategy is denoted as a vector of several *sorting criteria*, such as [criterion$_1$, criterion$_2$, ..., criterion$_k$], which defines a lexicographic ordering, i.e., from the OPEN list, first, select a set of nodes using criterion$_1$, and if there are still multiple nodes remaining in the set, then break ties using criterion$_2$ and so on, until a single node is selected. The *first-level sorting criterion* of a strategy is criterion$_1$, the *second-level sorting criterion* is criterion$_2$, and so on.[1]

[1]This notation corresponds to the command line option format of the Fast Downward planner

Using this notation, A^* without any tie-breaking strategy can be denoted as a BFS with $[f]$ and A^* which breaks ties according to h value is denoted as $[f, h]$. Unless stated otherwise, we assume the nodes are sorted in the increasing order of the key value and a BFS always selects the smallest key value.

However, a sorting strategy may only provide a partial ordering, i.e., the sorting strategy may fail to select a single node because some nodes may share the same sorting keys. For such cases, a BFS algorithm must decide which node to expand by applying some *default* tie-breaking criterion criterion_k which is guaranteed to return a single node, such as *fifo* (oldest node first: first-in-first-out), *lifo* (most recently inserted first: last-in-first-out) or *ro* (random ordering). For example, A^* using h tie-breaking and *fifo* default tie-breaking is denoted as $[f, h, \textit{fifo}]$. By definition, there is only 1 node which satisfies the default criterion, so strategies with a default criterion guarantee a total ordering among all nodes and are able to select a single node from the set of nodes. When the default criterion is irrelevant to the discussion, we either use a wildcard "*", e.g. $[f, h, *]$, or sometimes omit it altogether for brevity.

Given a search algorithm with a sorting strategy, a *plateau* (criterion ...) is a set of nodes in OPEN whose elements share the same sort keys according to non-default sorting criteria and are therefore indistinguishable. In the case of A^* using tie-breaking with h (sorting strategy $[f, h, *]$), the plateaus are denoted as *plateau* (f, h), the set of nodes with the same f cost and the same h cost. We can also refer to a specific plateau with $f = f_p$ and $h = h_p$ by *plateau* (f_p, h_p).

An *entrance* to a *plateau* (criterion ...) $= P$ is a node $n \in P$, whose current parent is not in P. The *final plateau* is the plateau containing the solution found

(Helmert, 2006).

by the search algorithm. In A^* using admissible heuristics, the final plateau is *plateau* (f^*) (without tie-breaking), or *plateau* $(f^*, 0)$ (with h-based tie-breaking).

Finally, OPEN list *alternation* (Röger & Helmert, 2010) is a technique to combine multiple sorting strategies in order to improve the robustness of the search algorithm. Nodes are simultaneously stored and sorted into independent OPEN lists with different strategies, and node expansion alternates among the OPEN lists. We denote an alternating OPEN list as *alt*$(X_1, X_2, \ldots)$ where each X_i is a sorting strategy.

Chapter 3

Analysis of Tie-Breaking Strategies and Plateau Structure for Cost-Optimal A*

In this chapter, we investigate *tie-breaking strategies* and *plateau structure* for cost-optimal A^* search. A^* is a standard search algorithm for finding an optimal cost path from an initial state s to some goal state $g \in G$ in a search space represented as a graph (Hart et al., 1968). It expands the nodes in best-first order of $f(n)$ up to f^*, where $f(n)$ is a lower bound of the cost of the shortest path that contains a node n and f^* is the cost of the optimal path. In many combinatorial search problems, the size of the last layer $f(n) = f^*$ of the search, called a *final plateau*, accounts for a significant fraction of the effective search space of A^*. Figure 3.0.1 (p.12) compares the number of states in this final plateau with $f(n) = f^*$ (y-axis) vs. $f(n) \leq f^*$ (x-axis) for 1104 problem instances from the International Planning Competition (IPC1998-2011). For many instances, a large

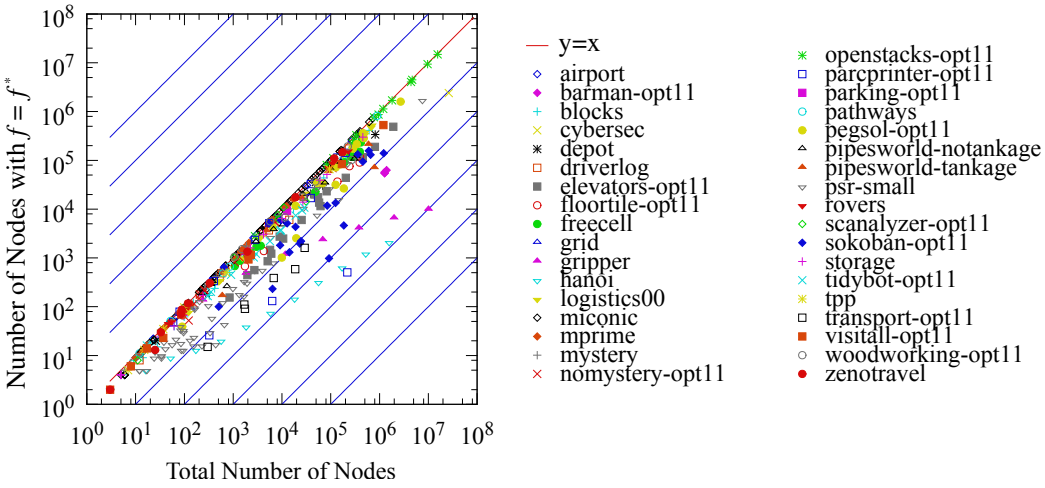

Figure 3.0.1: The number of nodes with $f = f^*$ (y-axis) compared to the total number of nodes in the search space (x-axis) with $f \leq f^*$ on 1104 IPC benchmark problems. This experiment uses a modified Fast Downward with LMcut which continues the search within the current f after any cost-optimal solution is found. This effectively generates all nodes with cost f^*.

fraction of the nodes in the effective search space have $f(n) = f^*$: The points are located very close to the diagonal line ($x = y$), indicating that almost all states with $f(n) \leq f^*$ have cost f^*.

Figure 3.0.2 depicts this phenomenon conceptually. On the left, we show one natural view of the search space that considers the space searched by A^* as a large number of closed nodes with $f < f^*$, surrounded by a thin layer of final plateau $f(n) = f^*$. This intuitive view accurately reflects the search spaces of some real-world problems such as 2D pathfinding on an explicit graph. It has also served as a model for algorithms such as Frontier Search (Korf, 1999; Korf &

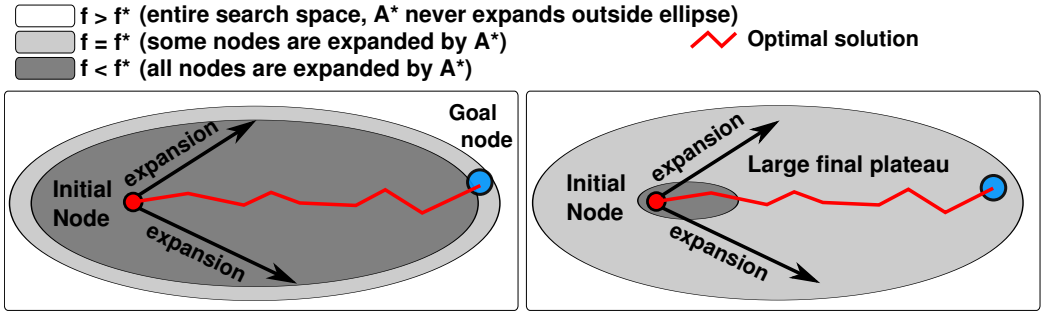

Figure 3.0.2: (Left) One possible class of search space which is dominated by the states with cost $f < f^*$. (Right) This thesis focuses on another class of search space, where the plateau containing the cost-optimal goals ($f = f^*$) is large, and it even accounts for most of the search effort required by A^*.

Zhang, 2000), which tries to reduce the memory requirement by discarding the information associated with states with $f < f^*$, an effective strategy when the number of such states accounts for a large fraction of the memory usage.

However, for many other classes of combinatorial search problems, e.g., the IPC Planning Competition Benchmarks, the figure on the right is a more accurate depiction – here, the search space has a large plateau for $f = f^*$. In fact, Iterative Deepening approaches (Korf, 1985) assume this type of search space where this final frontier is quite large and the overhead of re-evaluating $f < f^*$ is limited. Classical planning problems in the IPC benchmark set are clearly the instances of such combinatorial search problems.

For the majority of such IPC problem domains where the last layer ($f(n) = f^*$) accounts for a significant fraction of the effective search space, a *tie-breaking strategy*, which determines which node to expand among nodes with the same f-

cost, can have a significant impact on the performance of A^*. It is widely believed that among nodes with the same f-cost, ties should be broken according to $h(n)$, i.e., nodes with smaller h-values should be expanded first. While this is a useful rule of thumb in many domains, it turns out that tie-breaking requires more careful consideration, particularly for problems where most or all of the nodes in the last layer have the same h-value.

We empirically evaluate the existing, commonly used, standard tie-breaking strategies for A^* (Section 3.2). We show that:

1. In the experiments on IPC domains, A Last-In-First-Out (*lifo*) criterion tends to be more efficient than a First-In-First-Out (*fifo*) criterion.

2. Tie-breaking according to the heuristic value h, which is frequently mentioned in the heuristic search literature, has little impact on the performance as long as *lifo* default criterion is used – in other words, a *lifo* tie-breaking policy is sufficient for most IPC domains.

3. There are significant performance differences among tie-breaking strategies when domains include 0-cost actions. This is true even when h-based tie-breaking is used.

3.1 Tie-Breaking Strategies for A^*

A^* is a standard search algorithm for finding an optimal cost path on a graph. On a finite graph, A^* is complete regardless of the tiebreaking strategy (Hart et al., 1968).

It can be defined as a subclass of BFS which uses f-value as the first sorting criterion and returns a cost-optimal solution when h is admissible, i.e., when $\forall n; h(n) \leq h^*(n)$, where $h^*(n)$ is the optimal distance from n to the nearest goal. The best-first order of the expansion is the key to guaranteeing solution optimality. The first solution found by the algorithm is guaranteed to have the optimal cost $f = f^*$ because all nodes with $f(n) < k$ are already expanded when it starts expanding the nodes with $f(n) = k$. Thus, the *effective search space of* A^* is the set of nodes with $f(n) \leq f^*$: A^* expands all nodes with $f(n) < f^*$, then expands *some* of the nodes with $f(n) = f^*$, and never expands the nodes with $f(n) > f^*$.

If there are multiple nodes with the same f-cost, A^* must implement some tie-breaking strategy (either explicitly or implicitly) which selects from among these nodes. The early literature on heuristic search seems to have been mostly agnostic regarding tie-breaking. The original A^* paper, as well as Nilsson's subsequent textbook states: "Select the open node n whose value f is smallest. Resolve ties arbitrarily, but always in favor of any [goal node]" (Hart et al., 1968, p. 102 Step 2; Nilsson, 1971, p. 69). Pearl's textbook on heuristic search specifies that best-first search should "break ties arbitrarily" (Pearl, 1984, p. 48, Step 3), and does not specifically mention tie-breaking for A^*. To the best of our knowledge, the first explicit mention of a tie-breaking strategy that considers node generation order is by Korf in his analysis of IDA*: "If A^* employs the tie-breaking rule of 'most-recently generated', it must also expand the same nodes [as IDA*]", i.e., a *lifo* ordering.

In recent years, tie-breaking according to h-values has become "folklore" in the search community. Hansen and Zhou state that "[i]t is well-known that A^* achieves best performance when it breaks ties in favor of nodes with least h-cost"

(Hansen & Zhou, 2007). Holte writes "A^* breaks ties in favor of larger g-values, as is most often done" (Holte, 2010). Note that preferring large g is equivalent to preferring smaller h, since $f = g + h$. Felner et al. also assume "ties are broken in favor of low h-values" in describing Bidirectional Pathmax for A^* (2011). In their detailed survey/tutorial on efficient A^* implementations, Burns et al. (2012) also break ties "preferring high g" (equivalent to low h). Thus, tie-breaking according to h-values appears to be ubiquitous in practice. However, to our knowledge, an in-depth experimental analysis of tie-breaking strategies for A^* is lacking in the literature.

Although the standard practice of tie-breaking according to h might be sufficient in some domains, further levels of tie-breaking (explicit or implicit) are required if multiple nodes have the same f as well as the same h values. To date, the effect of such *default* tie-breaking has not been investigated in depth. For example, although the survey of efficient A^* implementation techniques by Burns et al. did not explicitly mention the default tie-breaking (2012), their library code uses *lifo* default tie-breaking (Burns, 2012). It first breaks ties according to h, and then breaks remaining ties according to a *lifo* criterion (most recently generated nodes first), i.e., $[f, h, \textit{lifo}]$. Although not documented, their choice of a *lifo* 2nd-level tie-breaking criterion appears to be a natural consequence of the fact it can be trivially and efficiently implemented in their two-level bucket (vector) implementation of OPEN. In contrast, the current implementation of the State-of-the-Art A^* based planner Fast Downward (Helmert, 2006), as well as the work by Röger and Helmert (2010) uses a $[f, h, \textit{fifo}]$ tie-breaking strategy. Although we could not find a published explanation, this choice is most likely due to their use of alternating OPEN lists, in which case the *fifo* second-level criterion serves to

provide a limited form of fairness.

3.2 Analysis of Standard Strategies

We first evaluated standard tie-breaking strategies for domain-independent cost-optimal classical planning and analyze their performance differences. In our experiments, all planners are based on Fast Downward, and all experiments are run with a 5-minute, 4GB memory limit for the search binary (FD translation/preprocessing times are not included in the 5-minute limit). All experiments were conducted on Xeon E5410@2.33GHz CPUs. For the randomized configurations, we took the average of 10 runs. We used two State-of-the-Art heuristic functions LMcut (Helmert & Domshlak, 2009) and M&S (Helmert et al., 2014) as the primary heuristic functions used for calculating f and h. For M&S, we used the bisimulation-based shrink strategy, DFP merge strategy, and exact label reduction. These basic experimental configurations are shared in all performance evaluation experiments throughout this chapter.

We used 1104 instances from 35 standard IPC benchmark domains: airport (50 instances), barman-opt11 (20), blocks (35), cybersec (19), depot (22), driverlog (20), elevators-opt11 (20), floortile-opt11 (20), freecell (80), grid (5), gripper (20), hanoi (30), logistics00 (28), miconic (150), mprime (35), mystery (30), nomystery-opt11 (20), openstacks-opt11 (20), parcprinter-opt11 (20), parking-opt11 (20), pathways (30), pegsol-opt11 (20), pipesworld-notankage(50), pipesworld-tankage (50), psr-small (50), rovers (40), scanalyzer-opt11 (20), sokoban-opt11 (20), storage (30), tidybot-opt11 (20), tpp (30), transport-opt11 (20), visitall-opt11 (20), woodworking-opt11 (20), zenotravel (20).

3.2.1 Is h-Based Tie-Breaking Necessary?

As noted in Section 3.1, the current standard practice is to use a tie-breaking criterion which uses the h-value of the nodes. However, to our knowledge, the need for h-based tie-breaking has not been previously empirically investigated.

In Table 3.2.1, we show the summary results for $[f, \textit{fifo}]$ and $[f, \textit{lifo}]$, the A^* variants which rely on *fifo* or *lifo* default tie-breaking only, as well as the standard $[f, h, \textit{fifo}]$ and $[f, h, \textit{lifo}]$ strategies. (Detailed results are in Table 9.1.1 and Table 9.1.2 in the Appendix.) $[f, \textit{lifo}]$, which simply breaks ties among nodes with the same f-cost by expanding the most recently generated nodes first (Korf, 1985), clearly dominates $[f, \textit{fifo}]$. Interestingly, the performance of the $[f, \textit{lifo}]$ strategy is comparable to $[f, h, \textit{lifo}]$ and $[f, h, \textit{fifo}]$. This may be surprising, considering the ubiquity of h-based tie-breaking in the search and planning communities.

This is explained by the fact that *lifo* behaves somewhat similarly to h-based tie-breaking. *lifo* expands the most recently generated node n. For any child n', if the heuristic function is admissible and $f(n') = f(n)$, there are only 2 possibilities : (1) $g(n') > g(n)$ and $h(n') < h(n)$, or (2) $g(n') = g(n)$ and $h(n') = h(n)$. Thus, as *lifo* expands nodes in a "depth-first" manner, the nodes that continue to be expanded in *plateau* (f) by *lifo* usually have non-increasing h-values, much like in h-based tie-breaking which always searches toward the least h cost. Thus, although the expansion order of $[f, \textit{lifo}]$ is not exactly the same as that of h-based tie-breaking strategies, they perform similarly.

Sorting Criteria	IPC(1104) LMcut	IPC(1104) M&S
$[f, \textit{fifo}]$	443	460
$[f, \textit{lifo}]$	558	490
$[f, \textit{ro}]$	448.9 ± 1.3	460.9 ± 1.6
$[f, h, \textit{fifo}]$	558	491
$[f, h, \textit{lifo}]$	**565**	**496**
$[f, h, \textit{ro}]$	558.9 ± 2.1	489.4 ± 1.0

Table 3.2.1: Summary of coverage comparison (5min, 4GB, LMcut heuristics) among the standard baseline tie-breaking algorithms (details in Table 9.1.1 and Table 9.1.2, leftmost 2 columns).

3.2.2 Do Default Strategies Make a Difference?

Next, we compared two commonly used tie-breaking strategies, $[f, h, \textit{fifo}]$, $[f, h, \textit{lifo}]$, which first break ties according to h, and then apply *fifo* or *lifo* default tie-breaking, respectively. Summary results for LMcut and M&S are shown in Table 3.2.1, and the detailed results are in Table 9.1.1 and Table 9.1.2 (Chapter 9, Appendix). Differences in coverage are observed in several domains and $[f, h, \textit{lifo}]$ outperforms $[f, h, \textit{fifo}]$ overall. Thus, the choice of default criterion seems to have a modest but measurable impact when the first tie-breaking criterion is h.

We also conducted experiments using *ro* (Random Order) default tie-breaking because it is another trivial way to break ties. We ran the experiments 10 times with the different random seeds, then took the average and the standard deviation of the coverages. The performance of *ro* is comparable to *fifo* default tie-breaking regardless of the primary heuristics, or the presence of h-based tie-breaking.

3.2.3 Plateaus and Tie-Breaking

Figure 3.2.1 provides a more fine-grained analysis by comparing the number of node evaluations (calls to the expensive LMcut heuristic function) on each instance by the $[f, h, \textit{lifo}]$ and $[f, h, \textit{fifo}]$ strategies. The difference in the number of nodes evaluated can sometimes be larger than a factor of 10 ($\mathrm{Openstacks}$, $\mathrm{Cybersec}$ domains). As noted in Section 3.1, the choice among default criteria has not been considered very important in the literature, as evidenced by the lack of explicit descriptions of the default tie-breaking criterion in recent papers. Our results suggest that 2nd-level default tie-breaking can have a surprisingly large effect on the search performance.

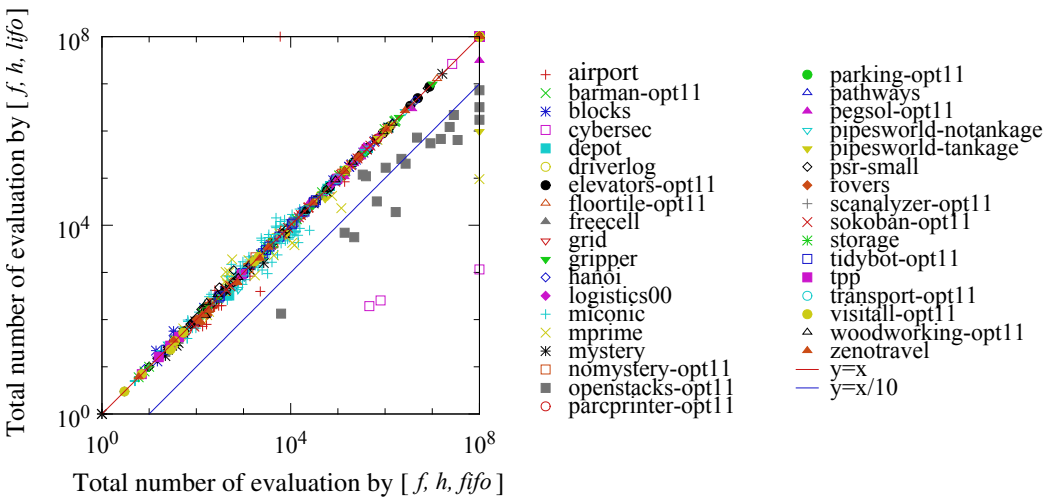

Figure 3.2.1: The number of LMcut evaluations on various IPC planning benchmark domains, with standard *fifo* vs *lifo* default tie-breaking, both with h tie-breaking. *lifo* evaluates less than $1/10$ of the nodes evaluated by *fifo* in Cybersec and Openstacks.

The effect of the choice of 2nd-level default tie-breaking criteria (*lifo* vs. *fifo*) when the 1st-level tie-breaking criterion is h tie-breaking is limited to each search plateau *plateau* (f, h), the set of nodes which share the same f value and h value. Also, in admissible search, two A^* implementations using different default tie-breaking criteria both expand the same set of nodes in the region where $f < f^*$. Furthermore, nodes with $h > 0$ can not be goal nodes when h is admissible. Therefore, the effect of default tie-breaking becomes most prominent in the final plateau, *plateau* $(f^*, 0)$.

Counterintuitively, the *plateau* $(f^*, 0)$ region can be large enough to cause a significant performance difference – in fact, this final plateau can even account for *most* of the search effort required by A^*. Figure 3.2.2 plots the size of the final plateau on 1104 IPC benchmark instances. The y-axis represents the number of nodes in the final plateau (*plateau* $(f^*, 0)$), and the x-axis represents the total number of nodes expanded so far. This figure suggests that in some domains such as Openstacks and Cybersec, the planner spends most of the runtime searching *plateau* $(f^*, 0)$ for a solution, even with the help of h tie-breaking.

A natural question is: What makes these two domains (Openstacks and Cybersec) different from all other domains which have much smaller final plateaus?

3.3 Domains with 0-Cost Actions

Openstacks is a cost minimization domain introduced in IPC-2006, where the objective is to minimize the number of stacks used. One characteristic of Openstacks is the presence of many actions which have zero cost because they do not increase the number of stacks. These 0-cost actions create the problem depicted in Fig-

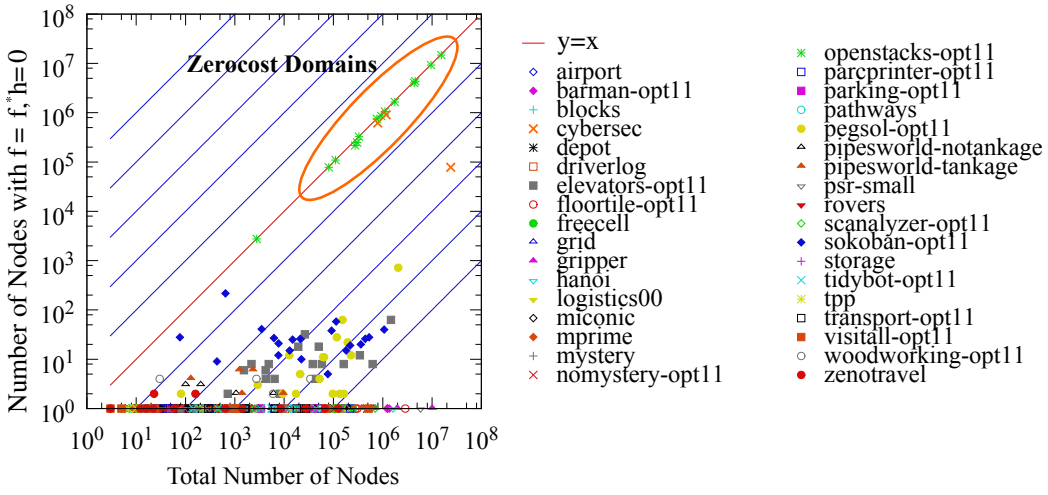

Figure 3.2.2: The number of nodes in *plateau* $(f^*, 0)$ (y-axis), which form the final plateau for sorting strategy $[f, h]$, compared to the total number of nodes in the search space with $f \leq f^*$ (x-axis) on 1104 IPC benchmark problems. Note that Openstacks and Cybersec instances are near the $y = x$ line. These statistics are obtained by running a modified Fast Downward with LMcut which continues searching after the solution is found until all nodes with cost $f = f^*$ are expanded.

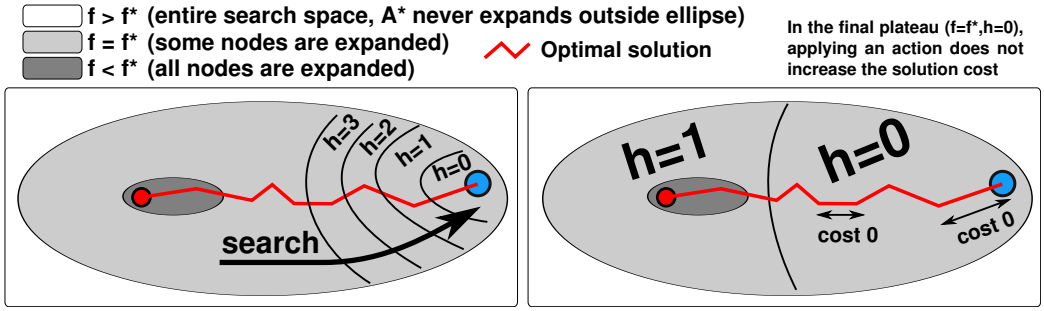

Figure 3.3.1: Search space of A^* and its contour according to admissible heuristic h. (**Right**) In domains with only positive-cost actions, h-based tie-breaking provides meaningful guidance. (**Left**) In domains with 0-cost actions, applying an action may not increase the cost of the path and the region with $h = 0$ could be quite large. With the same mechanism, other heuristic plateaus (e.g. $h = 1$) also become larger. Thus, h-based tie-breaking fails to provide meaningful guidance in this space.

ure 3.3.1. Since 0-cost actions (edges) allow "free" transitions between many neighboring nodes, the number of neighboring nodes sharing the same h also becomes quite large. This creates huge plateaus that share the same h-value, and the standard h-based tie-breaking criterion can not provide informative guidance for search within a plateau. Since the g-values of the nodes in these plateaus are all identical, these plateaus are an instance of g-value plateaus, which are known to increase the difficulty of search (Benton, Talamadupula, Eyerich, Mattmüller, & Kambhampati, 2010).

Although most traditional benchmark problems in the planning community and the combinatorial search community do not have 0-cost actions, we argue

that such domains are of an important class of models for cost-minimization problems, i.e., assigning 0-costs makes sense from a practical, modeling perspective. For example, consider the driverlog domain, where the task is to move packages between locations using trucks. The IPC version of this domain assigns unit costs to all actions. Thus, cost-optimal planning on this domain seeks to minimize the number of steps in the plan. However, another natural objective function would be the one which minimizes the amount of fuel spent by driving the trucks, assigning cost 0 to all actions except drive-truck – we believe that for cost-optimal planning, this is at least as natural as the current IPC model of driverlog in which all actions are of unit cost.

Similarly, for many practical applications, a natural objective is to optimize the usage of one key consumable resource, e.g., fuel/energy minimization. In fact, two of the IPC domains, Openstacks and Cybersec, which were shown to be difficult for standard tie-breaking methods in the previous section, both contain many 0-cost actions and are based on industrial applications: Openstacks models production planning (Fink & Voss, 1999) and Cybersec models Behavioral Adversary Modeling System (Boddy, Gohde, Haigh, & Harp, 2005, minimizing decryption, data transfer, etc.).

Therefore we modified various standard domains into cost minimization domains with many 0-cost actions. Specifically, each of our "Zerocost domains" is a standard domain which has been modified so that all action schema are assigned cost 0 except for a few (usually one) action schema which consumes some key resource. The suffixes in the names of these domains indicate the actions with non-zero costs, e.g., logistics-fuel is a modified logistics domain where only actions which consume fuel have non-zero cost. Most of the transportation-type do-

mains are modified to optimize energy usage (logistics-fuel, elevator-up etc.), and assembly-type domains are modified to minimize resource usage (woodworking-cut minimizes wood usage, etc.). When no action makes sense from the practical point of view, we chose an action schema arbitrarily (e.g. mprime-succumb). We did not include domains which have only a single action schema, or which already had many 0-cost actions.

The new set of 28 *Zerocost domains* are: airport-fuel (20 instances), blocks-stack (20), depot-fuel (22), driverlog-fuel (20), elevators-up (20), floortile-ink (20), freecell-move (20), grid-fuel (5), gripper-move (20), hiking-fuel (20), logistics00-fuel (28), miconic-up (30), mprime-succumb (35), mystery-feast (20), nomystery-fuel (20), parking-movecc (20), pathways-fuel (30), pipesnt-pushstart (20), pipesworld-pushend (20), psr-small-open (20), rovers-fuel (40), scanalyzer-analyze (20), soko-ban-pushgoal (20), storage-lift (20), tidybot-motion (20), tpp-fuel (30), woodworking-cut (20), zenotravel-fuel (20).

While the action costs in the PDDL *domain* definitions are modified, we did not modify the PDDL *problem* definitions. Although some domains (specifically, blocks, freecell, pipesworld-notankage, miconic) have fewer instances than the original domain does, their problem definitions are the evenly sampled subset of the original set of instances. For example, the original miconic domain has 150 instances, while our version has 30 instances. These 30 instances are selected evenly from the original set of instances, by picking instances p05, p10, ... p150. The reason for reducing the number of instances is to avoid the problem of the overall coverage sums being skewed by the domains with a larger number of instances. Thus, we did not modify the problem definitions at all, and only modified the action costs in the domain definitions.

3.3.1 Difference in Problem Characteristics between IPC and Zerocost Domains

Domains containing 0-cost operators are known to be difficult for traditional planners (Thayer & Ruml, 2009; Cushing, Benton, & Kambhampati, 2010; Wilt & Ruml, 2011; Thayer & Ruml, 2011; Richter, Westphal, & Helmert, 2011). Cushing et al. (2010) and Wilt and Ruml (2011) noted that a large ratio between maximum and minimum operator costs can pose a challenge to existing planners. They both addressed this using plan-length heuristics instead of plan-cost heuristics, which sacrifice the optimality of the solution. In contrast, we investigate methods for handling 0-cost operators within the framework of admissible search. In Section 5.3, we show how plan length heuristics can be incorporated into admissible search. In a parameterized complexity analysis of planning domains, Aghighi and Bäckström (2015, 2016) showed that domains with 0-cost operators comprise a complexity class that is harder (para-**NP**-hard) than the domains with strictly positive-cost operators (**W**[2] complete), indicating the inherent difficulty of optimally solving planning problems with 0-cost actions.

Therefore we experimentally evaluate whether our new set of Zerocost benchmarks based on standard IPC domains pose a new challenge for standard tie-breaking strategies. Results using the LMcut heuristic are shown in Table 3.3.1. In each table, the left-hand side shows the results in the original domains and the right-hand side shows the results for the corresponding Zerocost domains.

We observed a significant performance difference between the original IPC domains and the Zerocost domains. The coverage in Zerocost domains was lower in 11 domains while more instances were solved in 5 domains. The coverage increase in some domains is not surprising, considering that 0-cost actions also

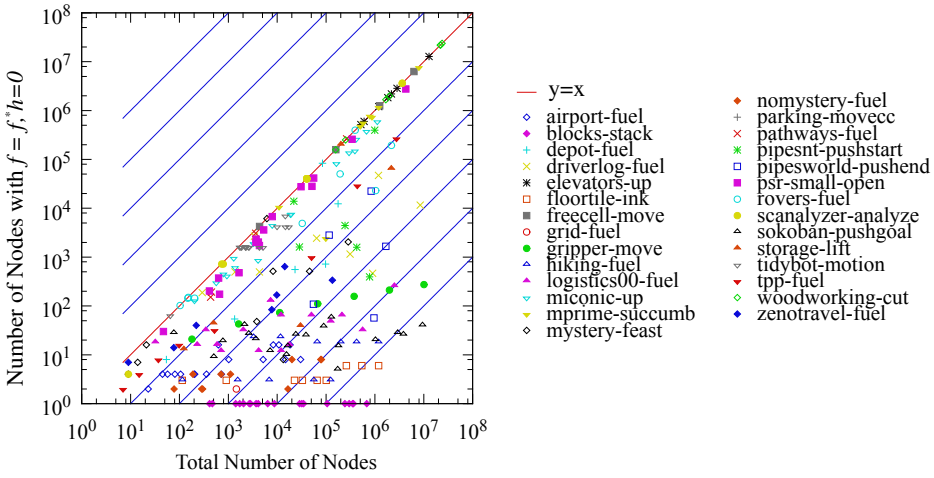

Figure 3.3.2: The number of nodes in *plateau* $(f^*, 0)$ (y-axis), which form the final plateau under h-based tie-breaking, compared to the total number of nodes in the search space (x-axis) with $f \leq f^*$ on 620 instances in our *Zerocost domains*. The final plateaus tends to account for a larger portion of the entire search space compared to Figure 3.2.2. These statistics are obtained by running a modified Fast Downward with LMcut which continues searching after the solution is found until expanding all nodes with cost $f = f^*$.

make some suboptimal paths into cost-optimal paths. However, the coverage decreased overall, confirming the difficulty of these domains.

Figure 3.3.2 plots the size of the final plateau of the Zerocost instances, with LMcut heuristics and h tie-breaking. In this plot, each point shows the total number of nodes in *plateau* $(f^*, 0)$ vs the total number of nodes with $f \leq f^*$. Compared to Figure 3.2.2, most Zerocost instances have larger plateaus even with the help of h tie-breaking. Thus, in these cost-minimization problems, the search strategy within plateaus, i.e., tie-breaking, becomes even more critical in determining search performance.

	solved	solved	(difference)	
depot(22)	6	6		depot-fuel(22)
driverlog(20)	13	8	(-5)	driverlog-fuel(20)
elevators-opt11(20)	15	7	(-8)	elevators-up(20)
floortile-opt11(20)	6	8	(+2)	floortile-ink(20)
grid(5)	1	1		grid-fuel(5)
gripper(20)	6	7	(+1)	gripper-move(20)
logistics00(28)	20	16	(-4)	logistics00-fuel(28)
mprime(35)	21	15	(-6)	mprime-succumb(35)
nomystery-opt11(20)	14	10	(-4)	nomystery-fuel(20)
parking-opt11(20)	1	0	(-1)	parking-movecc(20)
pathways(30)	5	5		pathways-fuel(30)
rovers(40)	7	8	(+1)	rovers-fuel(40)
scanalyzer-opt11(20)	10	9	(-1)	scanalyzer-analyze(20)
sokoban-opt11(20)	19	18	(-1)	sokoban-pushgoal(20)
storage(30)	14	4	(-10)	storage-lift(20)
tidybot-opt11(20)	12	16	(+4)	tidybot-motion(20)
tpp(30)	6	8	(+2)	tpp-fuel(30)
woodworking-opt11(20)	10	5	(-5)	woodworking-cut(20)
zenotravel(20)	11	7	(-4)	zenotravel-fuel(20)

Table 3.3.1: Assessment of the relative difficulty of Zerocost domains vs. their corresponding standard domains, for the standard $[f, h, \mathit{fifo}]$ strategy. Coverage comparison between the original IPC domains and the modified Zerocost domains are shown, using the same planner configuration and experimental setting (5min, 4GB, LMcut heuristics). This table does not include domains where the total number of instances in the Zerocost domain and the original domain differ.

Note that the difficulty posed by these domains sometimes *cannot* be tackled by improving the heuristic estimates, or reducing the underestimation of an admissible heuristic function. Due to the existence of 0-cost edges, some non-goal neighbors of a goal node have $h^* = 0$. For those nodes, there is clearly no room for improving the heuristic estimate; Any positive value causes the heuristics to be inadmissible.

One approach to improving the search performance in such plateaus produced by 0-cost edges is to perform an efficient *knowledge-free* search within plateau; It may reuse the effort that is already spent to guide the search but without requiring additional effort to compute multiple heuristics. In the next section, we propose and evaluate an implementation of such a technique. It turns out that introducing a notion of *depth* within a plateau can have a significant impact on the performance of knowledge-free search, and can also provide a good understanding of the behavior of standard tie-breaking strategies.

Chapter 4

Tiebreaking by Depth Diversification for A^* Search

As shown in the previous section, the search spaces of Zerocost domains have many 0-cost edges, resulting in a large final plateau (*plateau* $(f^*, 0)$). In a final plateau, all nodes have $h = 0$, so h-based tie-breaking cannot provide useful guidance toward a goal. Thus, we need a new metric for discriminating among nodes in the plateau so that the search algorithm can make progress on the plateau.

We define the *depth* of a node as an integer representing the distance (number of steps) from the *entrance* of the plateau. An *entrance* of the plateau is the first node which encountered the plateau along the path from the initial node. These notions are depicted in Figure 4.0.1 (subfigure 1).

The depth $d(n)$ of a node n is 0 when n and the parent node m are in the different plateaus, and $d(n) = d(m) + 1$ when they are on the same plateau. We omit (n) in $d(n)$ unless necessary, similarly to g and h for $g(n)$ and $h(n)$ (Chapter 2). As defined in Chapter 2, if two nodes are on the same plateau, they share

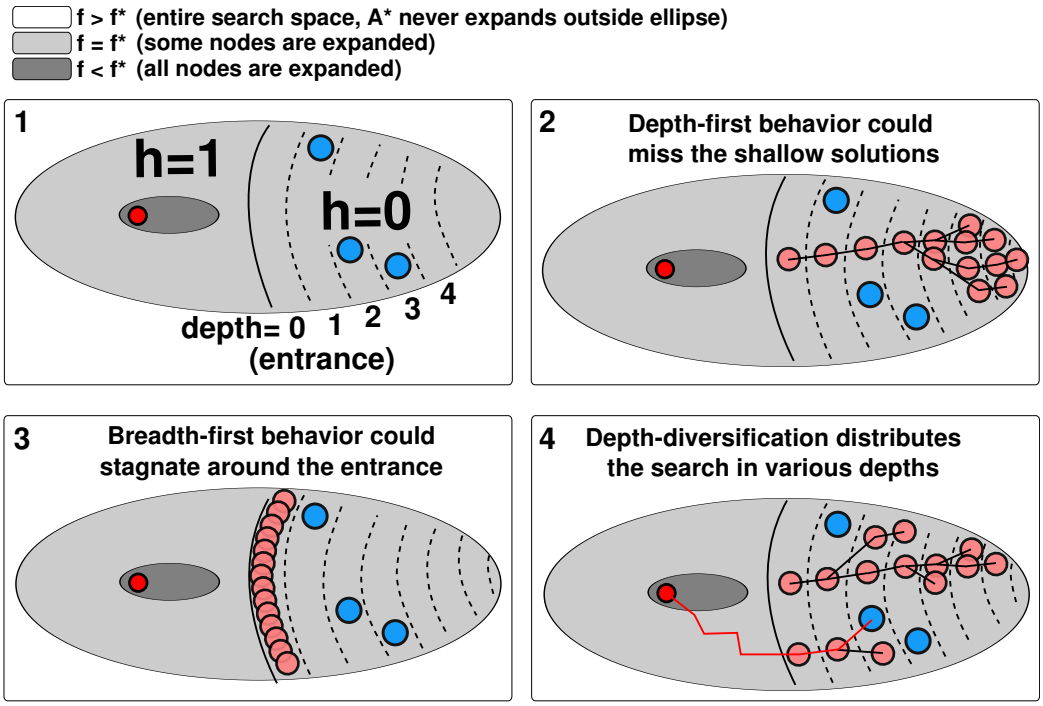

Figure 4.0.1: (**Subfigure 1**) The nodes in a plateau are divided into several layers, and each layer has a corresponding depth. Since all nodes have $f = f^*$, depth does not affect optimality, so all goals in the final plateau are cost-optimal, regardless of whether they are in shallow/deep regions. (**Subfigure 2**) *lifo* tie-breaking strategy results in depth-first behavior in a plateau, which could miss solutions if they are concentrated near the entrance. (**Subfigure 3**) *fifo* tie-breaking strategy results in breadth-first behavior in a plateau, which could fail to reach solutions in deeper layers within the time limit. (**Subfigure 4**) Depth-based diversification allows A^* to search the plateau space in a less biased manner. This balances exploration and exploitation, avoiding the problems with both *lifo* (depth-first) and *fifo* (breadth-first) behavior.

the same key values for the sorting strategy. For example, when the strategy is $[f, h, *]$, it means *plateau* $(f(n), h(n)) =$ *plateau* $(f(m), h(m))$, therefore $f(n) = f(m) \wedge h(n) = h(m)$.

The traditional *lifo* and *fifo* tie-breaking strategies search each plateau in the decreasing and the increasing order of the depth, respectively. Assume we are using $[f, h, *]$ sorting strategy. The *lifo* strategy always selects the most recently generated node within *plateau* (f, h), and the behavior in the plateau is equivalent to depth-first search. Thus, *lifo* always selects a node in the largest depth, as depicted in Figure 4.0.1 (subfigure 2). Similarly, the behavior of *fifo* strategy in a plateau is equivalent to breadth-first search. Thus *fifo* always selects the nodes with the least depth (subfigure 3). Note that $[f, h,$ *lifo*$]$ is equivalent to $[f, h, -d,$ *lifo*$]$ and $[f, h,$*fifo*$]$ is equivalent to $[f, h, d,$*fifo*$]$.

The problem with these traditional strategies is that we have no knowledge regarding whether the goals are located close to or far from the entrance. Recall that since $f = f^*$, all goal nodes in the final plateau are optimal with respect to solution cost regardless of the depth. However, until we find a solution, we do not know how the goals are distributed among various depths. In some problem instances the goals can be concentrated around the entrance, while in other problem instances the goals can be concentrated at some large depth.

In the former case, *fifo* should perform well because its breadth-first behavior naturally focuses the search around the entrance, favoring the smaller depths. However, in the latter case, exhaustively searching the shallower depths can result in not finding any solutions within the time limit because *fifo* may never reach the depth where the goals exist. On the other hand, *lifo* behaves in a depth-fist manner, so it may reach solutions at deeper depths quickly, but risks missing solutions at

shallower depths. Thus, both *fifo* and *lifo* tie-breaking are prone to failures due to pathological cases.

4.1 Depth-Based Tie-Breaking for A*

In order to avoid focusing the search at the wrong depths (too shallow/deep), the safest policy seems to be to simply *diversify* the depths which are being searched, in order to avoid any depth-based biases which could lead to pathological behavior. In our proposed *depth diversification* strategy, the nodes are inserted into buckets associated with depths, and upon expansion, search effort is distributed in a more balanced manner among various depths (Section 4.1.2 defines "more balanced" more precisely). Nodes are not "sorted" according to increasing or decreasing order of depth – instead, we try to "diversify" the node expansion within the plateau. We denote this depth diversification criterion as $\langle d \rangle$. For example, $[f, h, \langle d \rangle]$ first breaks ties according to h values, then uses the $\langle d \rangle$ criterion to break ties in *plateau* (f, h).

In order to diversify the expansion among depths, we simply iterate over the depth buckets (Algorithm 2). This iteration is managed by a Depth-Diversified Node Selector instance associated with each plateau (e.g. each of *plateau* $(1, 0)$, *plateau* $(2, 0)$, *plateau* $(2, 1) \ldots$). In order to select a single node from the OPEN list for expansion, we first select the plateau with the smallest key value, such as *plateau* $(f = 5, h = 1)$, as usual. This plateau is now represented by a selector instance, and we call pop(selector) method on this instance in order to obtain a node. Each instance holds an index d_c, the current depth (bucket index) selected in the last expansion, initialized to 0. On each call to pop(selector), the counter

Algorithm 2 Class Definition of Depth-Diversified Node Selector

Initialization of Instance Variables:

Counter $d_c \leftarrow 0$, Buckets $B = \{B_0, B_1, \ldots\}$, $\forall d; B_d = \emptyset$ (instantiated on-demand)

Method push(node n, selector):

Instantiate $B_{d(n)}$ if it does not exist

push$(n, B_{d(n)})$

Method pop(selector):

1: **loop**
2: $d_c \leftarrow d_c - 1$
3: $d_c \leftarrow |B| - 1$ **if** $d_c < 0$
4: **if** $B_{d_c} \neq \emptyset$ **then**
5: **return** pop(B_{d_c}) — Note: Actual "pop" method is subject to default tiebreaking.

is decremented ($d_c \leftarrow d_c - 1$) and a node is further popped from d_c-th bucket, which can be a *lifo*, *fifo* or *ro* queue. When d_c reaches below 0, then d_c is reset to the current largest depth in the plateau.

In an earlier, conference paper, we used a non-deterministic, randomized implementation of this idea (Asai & Fukunaga, 2016), which does not have this counter and pops a node from a randomly selected bucket ($B_{\texttt{random()}}$), but we use a deterministic implementation here because it facilitates the theoretical analysis below in Section 4.1.2.

Depth-based diversification is significantly different from the *ro* strategy which simply selects a random node from the OPEN list. The uniform sampling behavior of *ro* behaves very similar to *fifo*, and is insufficient to achieve the level of diversity provided by our depth diversification tie-breaking, which is also already evidenced by the performance similarity between *fifo* and *ro*-based tiebreaking strategies (Table 3.2.1). This is because at any given point in the search, more nodes will tend to have shallower depths than deeper depths, and a uniform, random selection will, therefore, be biased to select a node with shallow depths. For example, imagine we have 100 nodes at depth $d = 1$ and a single node at depth $d = 2$. Since *ro* does not consider the depth, the chance of expanding $d = 2$ is only $1/101$. This probability does not improve until a sufficient number of expansions decreases the number of nodes in $d = 1$. In contrast, our depth diversification policy expands nodes at $d = 1$ and $d = 2$ with equal probability.

Depth-based tie-breaking does not affect the order of node expansion when there are no remaining ties after the higher priority tie-breaking criteria, in which case all nodes have depth 0. More formally:

Lemma 1. *If all edge costs are positive, then* $\mathrm{d}(n) = 0$ *for every node* n *expanded*

by $A^*\ [f, h, \langle d \rangle, *]$.

Proof. Let n be a child of a node m. Regardless whether the parent m of the node n is newly assigned, updated, or the old parent is kept in line 10 of Algorithm 1, the invariant $g(n) = g(m) + \mathit{cost}(m, n) > g(m)$ holds because $\mathit{cost}(m, n) > 0$, and therefore $f(n) - h(n) > f(m) - h(m)$. This means that either $f(n) \neq f(m)$ or $h(n) \neq h(m)$, so $d(n) = 0$. □

Theorem 1. *If all edge costs are positive, then* $A^*\ [f, h, \langle d \rangle, *]$ *expands nodes in the same order as* $A^*\ [f, h, *]$ *(where "*$*$*" is any criterion).*

Proof. By Lemma 1, all nodes expanded by $A^*\ [f, h, \langle d \rangle, *]$ have depth 0, and all nodes are in the same depth bucket in Algorithm 2, so $A^*\ [f, h, \langle d \rangle, *]$ expands nodes in the same order as $A^*\ [f, h, *]$ regardless of the criterion $*$. □

4.1.1 Tie-Breaking within Depth Buckets

Depth diversification cannot be a default tie-breaking by itself. Consider a tie-breaking strategy such as $[f, h, \langle d \rangle]$ which applies a depth-diversification tie-breaking. After the $\langle d \rangle$ criterion is applied, there may be multiple nodes within the same depth bucket, so a default tie-breaking criterion is still necessary to break ties among them. Thus, we should, for example, apply one of *lifo*, *fifo* or *ro* (random order) criteria after the $\langle d \rangle$ criterion.

There are two concerns about this default tie-breaking criteria. First, the default tie-breaking behavior is still susceptible to accidental biases, e.g., names / orders of action schema in the PDDL domain definition (Vallati, Hutter, Chrpa, & McCluskey, 2015). Second, in addition to accidental biases, there may be some nontrivial biases that require sophisticated algorithms to be removed.

Domain Configuration and Tiebreaking

Recently, Vallati et al. showed that the performances of satisficing planners were significantly affected by PDDL domain *configurations*, which include the name / ordering of actions, propositions, and objects in the PDDL input file (2015). They conjectured that performance variations caused by different domain configurations are due to the impact that the naming/ordering of objects has on tiebreaking. In Fast Downward, action names can affect search performance, because FD sorts the action schemas according to the dictionary order of the schema names, which affects the order of applicable ground actions, which in turn affects the node insertion order into OPEN. We discuss this in Section 4.2.2.

Other Non-trivial Biases

In addition to accidental biases, there may be other nontrivial biases such as some form of symmetry among states which can be removed using some tie-breaking criterion X. Such a criterion can be applied after the depth criterion but before the default criterion, resulting in a sorting strategy $[f, h, \langle d \rangle, X, \textit{fifo}]$. Candidates for X may be related to pruning techniques such as Symmetry Breaking (Fox & Long, 1998; Pochter, Zohar, & Rosenschein, 2011; Domshlak, Katz, & Shleyfman, 2013) or Partial Order Reduction (Hall, Cohen, Burkett, & Klein, 2013; Wehrle, Helmert, Alkhazraji, & Mattmüller, 2013). While these are usually described as "pruning techniques", they can also be interpreted as strong bias removal mechanisms because they seek to prune redundant nodes, and redundancy causes a biased search effort. For example, imagine we have a set of nodes $S = \{a_1, a_2, a_3, a_4, b, c, d\}$ where $A = \{a_1, a_2, a_3, a_4\}$ are "redundant" according to some measure (e.g. by Symmetry, Partial-Order). If a search algorithm

expands S by random selection, it favors the group A by giving 4 times larger chance of expansion than each of b, c or d. Despite this similarity, search diversification is weaker than pruning methods because diversification can only *delay* the expansion of nodes sharing the similar attributes (such as depth), not *prune* the nodes.

4.1.2 Theoretical Characteristics of the Depth Distribution

We give further insight into the search behavior of our implementation of depth-based diversification. In depth-based diversification, although it is possible to select from a randomly selected depth bucket, as was done in an earlier conference paper (Asai & Fukunaga, 2016), the implementation used in this thesis performs a deterministic, round-robin sampling from the available depth buckets as described in Algorithm 2. We are particularly interested in how the nodes selected for expansion are distributed among the various depths in a plateau region. Assume that a search algorithm is searching a plateau region P. The precise definition of P depends on the higher-level sorting strategy e.g. $[f, h, \langle d \rangle]$ or $[f, \langle d \rangle]$. Using a simplified model where this P forms a forest (a set of disjoint trees), we can analyze the number of expansions in a particular depth can be represented by a simple formula.

In the discussion below, we first assume that P forms a forest of a fixed branching factor $w \geq 2$ (*forest assumption*), rather than a graph with an indefinite number of successor nodes. In the later experiments, we show this is a fairly accurate model. We also assume that no depth bucket is exhausted due to the expansion (*no-exhaustion assumption*). This implies that there are a sufficiently large number of nodes in depth $d = 0$ so that depth 0 is not exhausted, which may cause *fifo*

default tiebreaking to fail due to the heavy bias to the shallow depth. We provide a condition for this assumption to hold within this section. An example of running depth diversification with $w = 3$ is depicted in Figure 4.1.2.

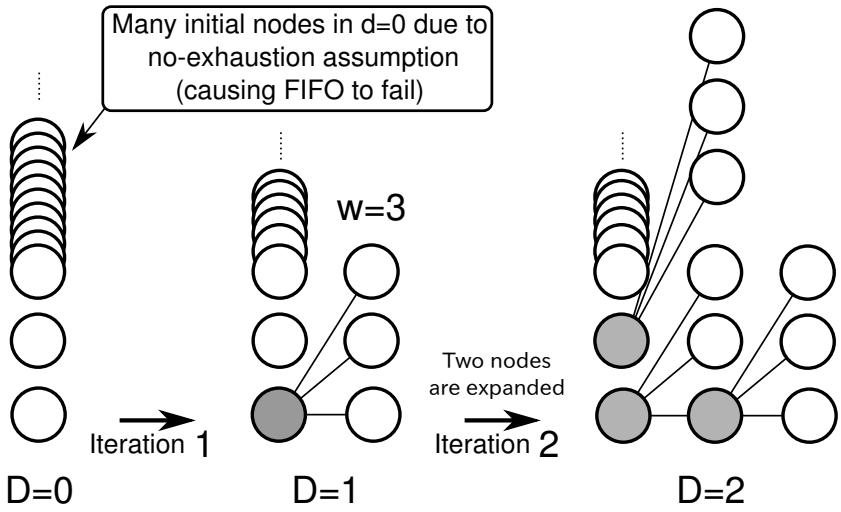

Figure 4.1.1: Depth Diversification applied to a plateau with forest assumption and no-exhaustion assumption.

Let $D \geq 0$ be the current largest depth of the nodes found in P so far. This is equal to $|B| - 1$ in Algorithm 2, the size of the buckets in Depth-Diversified Node Selector instance. An expansion of a node at depth D results in w more nodes with depth $D + 1$ on the same plateau P. These children are all newly generated because by the forest assumption, each child has a single incoming edge. Since the expansion is diversified by a sequence of iterations from the current largest depth to 0, when the current largest depth of the plateau is D, the number of iteration executed so far is also D because at the beginning of each iteration the largest depth is increased by 1. Therefore, at the end of the D'th iteration, each depth d has been expanded exactly $D - d$ times, with $D(D - 1)$ expansions in total. In

Figure 4.1.2, after iteration 2, depth $d = 0$ is expanded twice and depth $d = 1$ is expanded once.

It also means that a sufficient condition for *no-exhaustion assumption* to hold until the end of the D'th iteration is that the initial number of nodes in depth 0 is at least D. If there are at least D nodes in depth 0, depth 0 is trivially never exhausted until the D'th iteration. Also, no depth buckets in depth $d > 0$ will be exhausted because each bucket has $w(D - d + 1)$ generated nodes in total (i.e. OPEN+CLOSED) while the expansion has happened only $D - d$ times. The number of nodes in each bucket ($w(D - d + 1)$) follows from the fact that depth $d - 1$ is expanded $D - (d - 1)$ times in the preceding D iterations. Since $w \geq 2$, $w(D - d + 1) \geq 2(D - d) + 2 > D - d$.

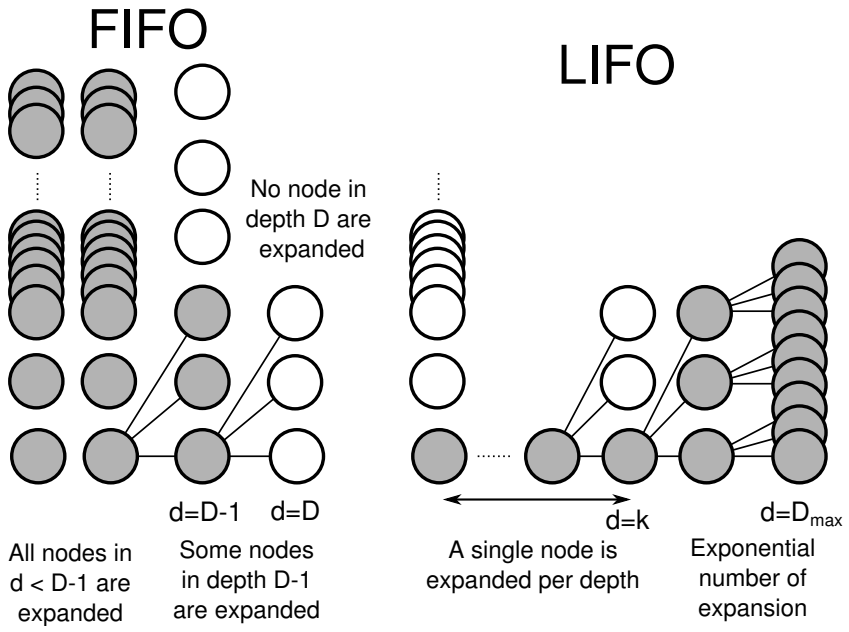

Figure 4.1.2: FIFO and LIFO applied to a plateau with forest assumption and no-exhaustion assumption.

If there are no solutions, every depth-selection criterion, including least depth selection (*fifo*) or largest depth selection (*lifo*), expands the same set of nodes and results in the same distribution as depth diversification. For example, if the number of nodes in depth 0 is D, each d is expanded Dw^d times. However, their online characteristics are different (Figure 4.1.2). Under our assumptions, the $D - d$ distribution of depth diversification is an invariant which holds at any point in the search until the solution is found. In contrast, in *fifo*, all nodes with $d < D - 1$ are expanded, depth $d = D - 1$ can take an arbitrary number of expansions $e \in [0, Dw^{D-1}]$ and $d \geq D$ are not expanded at all. In *lifo*, for some $k \in [0, D_{max}]$ (assuming the forest has a finite maximum depth D_{max}), there can be a situation where all depths $d \in [0, k]$ get only 1 expansion each while all nodes in depths $d \in [k + 1, D_{max}]$ are expanded. In this case, the number of expansions in $d \in [k, D_{max}]$ is exponential to $D_{max} - k$ ($\sum_{i=k}^{i=Dmax} w^{i-k} = \frac{1-w^{Dmax-k+1}}{1-w}$) while the number of expansions in $d \in [0, k - 1]$ is linear to k (i.e. $k - 1$). Such an imbalance during the search causes the pathological behavior mentioned above.

4.2 Evaluating Depth-Based Tie-Breaking

We compared the performance of standard tie-breaking methods to depth-based tie-breaking methods. These all use h as the second-level sorting criterion and either *fifo*, *lifo* or *ro* (random order) default tie-breaking criterion. The only difference is the presence of the third, depth-diversification criterion.

Experiments are conducted on 1104 standard IPC benchmark instances from 35 domains and 620 Zerocost instances from 28 domains (see Section 3.2 and Section 3.3 for full lists of these domains). The basic experimental settings are

the same as the previous ones: Each experiment uses the Fast Downward planner using A^* search and either the LMcut heuristic or M&S heuristic. Each experiment is run for 5 minutes excluding SAS translation time, with 4GB memory constraints.

We first show the summary results of these experiments (Table 4.2.1). Overall, depth-based tie-breaking tends to show larger coverages than the standard tie-breaking strategies. Interestingly, when the depth diversity criterion $\langle d \rangle$ is used, the performance relationship between *lifo* and *fifo* seems to flip: *fifo* tends to perform better than *lifo* in Zerocost domains for both LMcut and M&S heuristics (299 vs 279 for LMcut, 317 vs 303 for M&S). Also, *ro* (random order) outperforms both *fifo* and *lifo*. In the following, we describe and discuss each experiment. Detailed data tables are in the Appendix (Section 9).

Table 9.2.1 and Table 9.2.2 show the number of Zerocost instances (out of 620) solved by LMcut and M&S heuristics. In these Zerocost domains, our proposed method outperforms the traditional tie-breaking methods in both heuristics. Significant improvements were observed in 10 domains when using LMcut, and 7 domains when using M&S.

Table 9.2.3 shows the number standard IPC benchmark instances (out of 1104) solved by the configuration using LMcut heuristics. Depth-based tie-breaking ($\langle d \rangle$) achieves impressive results on $\mathrm{Openstacks}$ (*fifo* : $2 \rightarrow 8$, *lifo* : $3 \rightarrow 12$, *ro* : $3.9 \rightarrow 10$) and $\mathrm{Cybersec}$ (*fifo* : $11 \rightarrow 18$, *ro* : $11.7 \rightarrow 18$) because these domains contain many instances of 0-cost edges (See Figure 3.2.2). Most other instances are unaffected by depth-based tie-breaking. Thus, depth-based tie-breaking yields better performance in the domains with 0-cost actions, without sacrificing performance in other domains.

Sorting Criteria	Zerocost(620) LMcut	Zerocost(620) M&S	IPC(1104) LMcut	IPC(1104) M&S
Standard				
$[f, h, \textit{fifo}]$	256	280	558	491
$[f, h, \textit{lifo}]$	279	301	565	**496**
$[f, h, \textit{ro}]$	261.9 ± 1.4	287.7 ± 3.2	558.9 ± 2.1	489.4 ± 1.0
Depth-based				
$[f, h, \langle d \rangle, \textit{fifo}]$	284	302	571	487
$[f, h, \langle d \rangle, \textit{lifo}]$	264	288	**575**	487
$[f, h, \langle d \rangle, \textit{ro}]$	**288.1 ± 1.6**	**308.1 ± 2.1**	571.4 ± 1.7	485.6 ± 1.5

Table 4.2.1: Main summary results: Coverage comparison (number of instances solved in 5min, 4GB, LMcut/M&S heuristics) between standard tie-breaking and depth-based tie-breaking ($\langle d \rangle$). When LMcut is used, $\langle d \rangle$ outperforms standard strategies both in IPC instances (1104 problems total) and Zerocost instances (620 problems total). When M&S is used, $\langle d \rangle$ outperforms standard strategies in Zerocost instances. **Bold** shows the best configuration.

In contrast, Table 9.2.4 shows that depth-based tie-breaking degrades the performance of the configuration using M&S when applied to 1104 standard IPC benchmark instances. This result can be explained as follows. First, similar to the case of LMcut, $\mathrm{Openstacks}$ coverage improved for *fifo* ($15 \rightarrow 19$) and *ro* ($15.4 \rightarrow 19$), which is expected according to our analysis of Zerocost domains. Although there was no improvement on $\mathrm{Cybersec}$, this is because the coverage of $\mathrm{Cybersec}$ is 0 in all M&S configurations, regardless of tie-breaking. Thus, the

positive contribution of depth diversification to the overall score was limited for M&S compared to LMcut.

Second, with M&S, performance degraded across a wide range of domains due to the low-level overhead of depth-based tie-breaking (i.e., updates to the depth-based bucket data structures). As shown in Figure 4.2.1, when depth-based tie-breaking was used, the node evaluations rate significantly decreased with the M&S heuristic, while node evaluation rate decreased much less for LMcut. This is because the M&S heuristic is implemented as an efficient table lookup, and M&S is able to evaluate an order of magnitude larger number of nodes compared to LMcut. Thus, even the relatively small overhead incurred by depth bucket updates decreases the node evaluation rate enough to noticeably degrade M&S performance. Figure 4.2.2 shows a cumulative coverage plot which shows the number of node evaluations required to solve IPC instances. According to Figure 4.2.2, the number of evaluations required to solve IPC instances for $[f, h, *]$ and $[f, h, \langle d \rangle, *]$ were almost identical, which is expected because IPC instances mostly consist of instances with only positive-cost actions which are unaffected by depth-based tie-breaking (as predicted by our analysis in Section 4.1). This shows that the coverage degradation on IPC instances when using depth diversification is caused by the low-level overhead.

Finally, the per-domain results for Zerocost domains (Tables 9.2.1 - 9.2.2) show that $\langle d \rangle$ can cause both improvement and degradation (despite the total coverage improvement). This is natural considering that depth-diversification is designed to be a conservative, domain-independent strategy which is designed to avoid worst-case pathological behaviors. Overall, $\langle d \rangle$ tends to perform well, but the best-performing strategy on particular domain varies — for example, *fifo* is the

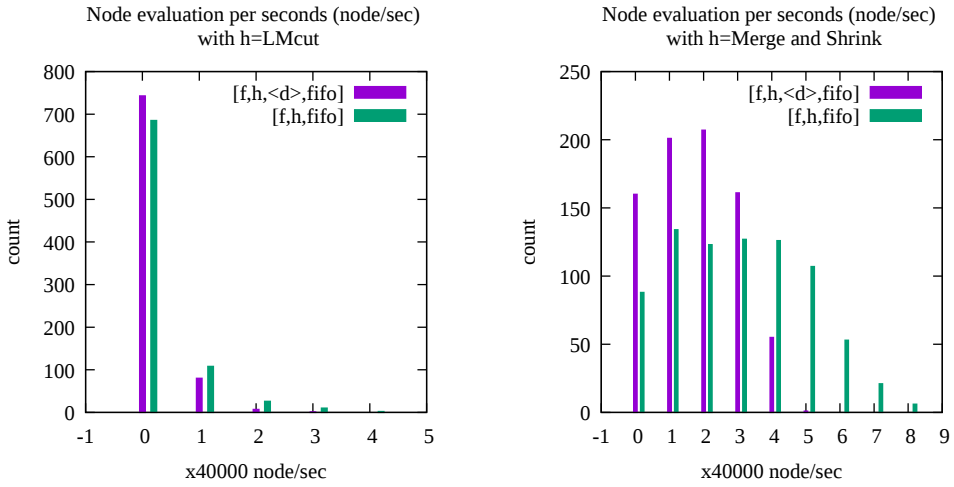

Figure 4.2.1: Histogram comparing the node evaluation ratio (node/sec) between standard tie-breaking ($[f, h, \mathit{fifo}]$) and depth-based tie-breaking ($[f, h, \langle d \rangle, \mathit{fifo}]$) on LMcut and M&S heuristics in IPC and Zerocost instances. x-axis shows the number of nodes expanded per seconds, and y-axis shows the number of problem instances. (See Appendix Figure 9.3.1 for the data on $[f, h, \mathit{lifo}]$ vs. $[f, h, \langle d \rangle, \mathit{lifo}]$.) On M&S, compared to LMcut, node evaluation rate more often becomes slower when depth is enabled. This is because the node evaluation of M&S is an order of magnitude faster than LMcut, and the overhead of managing depth-based tie-breaking queue becomes significant.

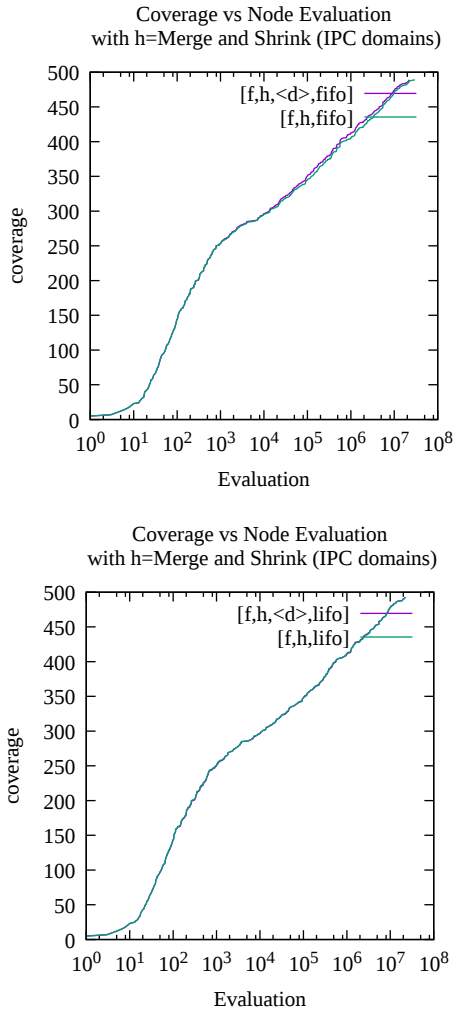

Figure 4.2.2: Cumulative coverage (y-axis) vs the number of evaluated nodes (x-axis), on IPC instances solved by both $[f, h, *]$ and $[f, h, \langle d \rangle, *]$ where $h =$ M&S. Left: *fifo*, Right: *lifo*.

best in airport-fuel with LMcut, while *lifo* is the best in freecell-move with LMcut. An adaptive tie-breaking which selects the tie-breaking strategy for a given domain is discussed in Section 5.5.4.

4.2.1 Search Behavior Within a Plateau

To understand the behavior of depth-based policies, we plotted histograms of the depths of search nodes evaluated by several tie-breaking strategies in the final plateau *plateau* $(f^*, 0)$ until the solution is found. We plotted a depth-based strategy $[f, h, \langle d \rangle, \textit{fifo}]$, as well as the standard strategies $[f, h, \textit{fifo}]$, $[f, h, \textit{lifo}]$ and a single run of randomized strategy $[f, h, \textit{ro}]$.

In order to obtain the data for the strategies which do not use depth-based tie-breaking ($[f, h, \textit{fifo}]$, $[f, h, \textit{lifo}]$, $[f, h, \textit{ro}]$), we added some instrumentation to these strategies so that, the depth of each of the expanded nodes is computed, although they do not affect the search behavior. Note that this instrumentation, which adds some runtime overhead, was *not* used in the performance comparison experiments above, and were only used for this experiment, which analyzes search behavior.

Figure 4.2.3 (as well as Figures 9.4.1 - 9.4.2 in the Appendix) show the results on exemplary instances from various Zerocost domains. We do not show some domains where we did not observe any depths greater than 3, in which case both the depth metric and *lifo*/*fifo*/*ro* have a negligible impact on search performance. We observed very similar results across a wide range of domains as shown in the figures. This indicates that the depth metric accurately describes the behavior of each tie-breaking criterion.

For example, consider the first figure, which plots depths searched on depot-fuel, p07. The $[f, h, \textit{lifo}]$ plot shows that the depth-first behavior results in deeper

search ($\approx 10^3$), while only a handful of nodes are expanded at intermediate depths (usually once). Thus, *lifo*'s depth-first behavior is prone to missing the key branch at intermediate depths that may lead to solutions earlier. On the other hand, the breadth-first behavior of $[f, h, \textit{fifo}]$ often gets stuck spending an excessive amount of time searching around the plateau entrance (expanding $\approx 10^3$ nodes at depth 10).

Also, we noticed that the node distribution of the global randomization $[f, h, \textit{ro}]$ is very similar to $[f, h, \textit{fifo}]$. This shows that *ro* actually behaves very similar to *fifo*, which is consistent with the previous performance comparisons in Section 3.2 and our observation regarding *ro* in Section 4.1. Thus, the overall behavior of *ro* tends to be similar to *fifo*, and naive randomization does not solve the problem of heavy bias for shallower depth nodes.

In contrast, $[f, h, \langle d \rangle, \textit{fifo}]$ is balancing the search at various depths. The yellow curve representing $[f, h, \langle d \rangle, \textit{fifo}]$ tends to be almost flat at shallow depths while gradually decreasing the number of nodes at larger depths. Moreover, its node distribution almost accurately follows $D - d$, a theoretical model from Section 4.1.2 which applies the simplified assumption that the plateau is a forest with a fixed branching factor. D denotes the largest depth of the unexpanded nodes in the final plateau, which is 1 larger than the largest depth of the expanded nodes.

The discrepancy of the $[f, h, \langle d \rangle, \textit{fifo}]$ curve from the theoretical prediction $D - d$ can be caused by the following factors: First, the outdegree of each node in the graph may not be uniform across the search space. Second, some depth buckets could be exhausted, as depicted in the $[f, h, \textit{fifo}]$ line which shows that all nodes in the shallower depths are expanded while the line is still below $D - d$. Since $[f, h, \textit{fifo}]$ exhaustively expands the nodes in shallower depth, the number of

expansion by $[f, h, \textit{fifo}]$ in the shallower depths constitutes an upper bound, which may be below $D - d$.

Next, Figure 4.2.4 shows the same results on the standard IPC $\mathrm{Openstacks}$ and $\mathrm{Cybersec}$ domains. The $\mathrm{Openstacks}$ results were similar to those of the Zerocost domains. In $\mathrm{Cybersec}$, we found that the performance improvement was not due to the number of nodes in *plateau* $(f^*, 0)$, because all tie-breaking strategies have generated only a small number of such nodes before the solution was found. Instead, we observed a large difference in the depth distributions in non-final plateaus *plateau* $(f^*, h), h \neq 0$ caused by the difference of tie-breaking. Note that depth diversification is always applied regardless of f or h values. This suggests that most children of the nodes in *plateau* (f^*, h) have f value larger than f^* or stays in *plateau* (f^*, h), and the planner is struggling to find nodes with better h. Due to the unbiased search, the depth-based strategy has a better chance of improving h values, finding a node in *plateau* $(f^*, 0)$ more quickly. This shows that considering depth can also help the search in non-final plateaus to find the nodes in the next plateau. Similar phenomena were observed in several other instances and domains, e.g., $\mathrm{depot\text{-}fuel}$, $\mathrm{driverlog\text{-}fuel}$, $\mathrm{zenotravel\text{-}fuel}$, $\mathrm{floortile\text{-}ink}$, $\mathrm{mprime\text{-}succumb}$, $\mathrm{storage\text{-}lift}$ (Figure 9.5.1 in Appendix).

Note that the small number of nodes in *plateau* $(f^*, 0)$ in this experiment does not contradict the results in Figure 3.2.2, which shows that the number of such nodes is quite large. This is because, while in Figure 3.2.2 the search continues until expanding all nodes in the final plateau, in this experiment the search stops when the first solution is found – Figure 3.2.2 was intended to show the size of the entire final plateau, while Figures 4.2.3 - 4.2.4 were meant to show the actual search behavior. If we continue the search until exhausting the final plateau, all

tie-breaking strategies will expand the same set of nodes (in different orders), so we would obtain plots similar to Figure 3.2.2 regardless of the tie-breaking strategy.

4.2.2 The Effect of Domain Mangling

We tested the robustness of the standard $[f, h, \textit{lifo}]$ and $[f, h, \textit{fifo}]$ strategies, as well as $[f, h, \langle d \rangle, \textit{ro}]$, with respect to biases introduced by domain configuration (action naming) in the PDDL domain definition. We created 3 different sets of domains in which the original names of action schema are mangled into random strings. We ran each of the 3 strategies on each set of mangled domains, three times each with different random seeds, resulting in 9 runs per strategy.

The results are shown in Table 4.2.2. We statistically analyzed the results for $[f, h, \langle d \rangle, \textit{ro}]$ to see if any of the 4 sets of domains significantly outperformed the others. Fligner-Killeen's non-parametric test could not reject the homogeneity of variances ($p = 0.75$ for IPC, $p = 0.26$ for Zerocost), so we then applied the non-parametric Kruskal-Wallis test, which showed that the mean differences were not significant ($p = 0.28$ for IPC, $p = 0.44$ for Zerocost), i.e., action name mangling did not significantly affect performance.

Thus, in contrast to the results for satisficing search by (Vallati et al., 2015), the effect of action ordering seems to be relatively weak for cost-optimal search using A^*. This may be because compared to the satisficing, best-first search algorithms evaluated in (Vallati et al., 2015), the behavior of admissible search is more constrained.

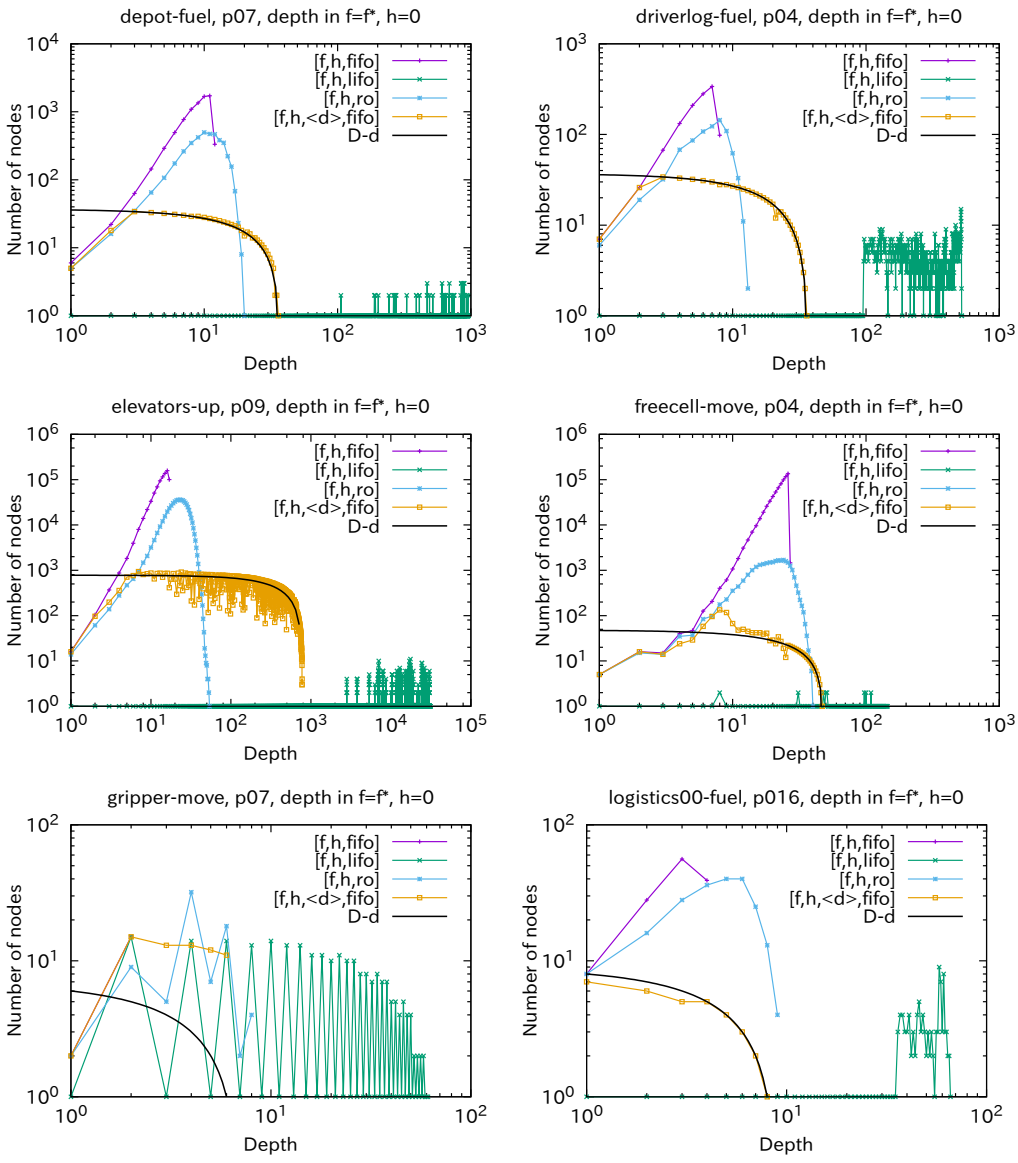

Figure 4.2.3: Number of nodes (y-axis) expanded per depth (x-axis) in the final plateau with different tie-breaking strategies. Both axes are in logarithmic scale.

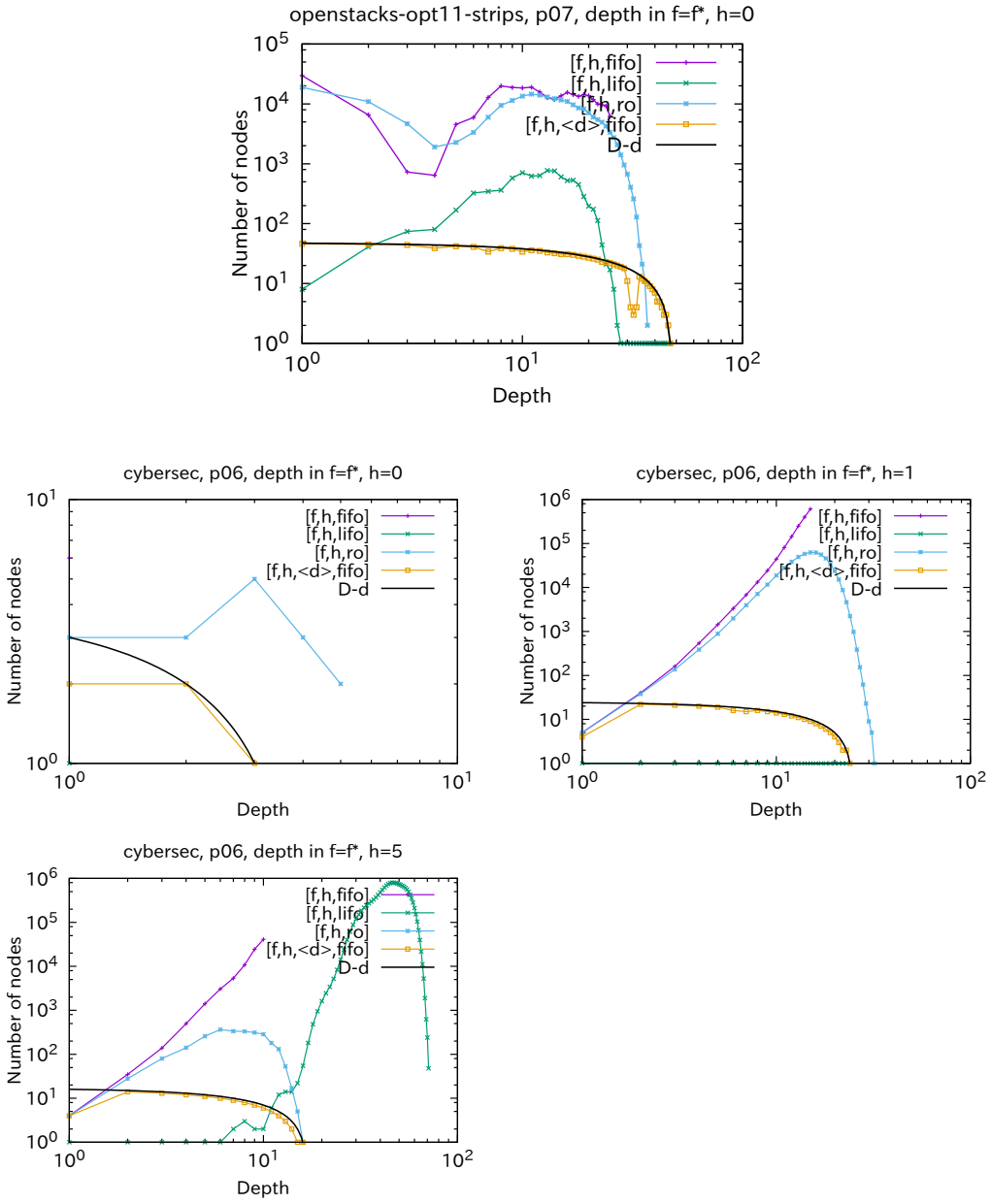

Figure 4.2.4: Depth distribution of Openstacks and Cybersec instances in the final (*plateau* $(f^*, 0)$) and non-final plateaus (*plateau* (f^*, h), $h \neq 0$). In Cybersec p06, although the number of nodes generated in *plateau* $(f^*, 0)$ is small, *fifo* and *ro* behaved poorly on *plateau* $(f^*, 1)$, and also *lifo* behaved poorly on *plateau* $(f^*, 5)$.

Domain	$[f, h, \textit{fifo}]$	$[f, h, \textit{lifo}]$	$[f, h, \langle d \rangle, \textit{ro}]$ (n: number of runs)
Mangled IPC 1 (1104)	556	564	571.7±0.9 ($n = 3$)
Mangled IPC 2 (1104)	557	568	571.3±0.9 ($n = 3$)
Mangled IPC 3 (1104)	557	568	573.0±1.6 ($n = 3$)
Original IPC (1104)	558	565	570.6±1.5 ($n = 10$)
Mangled Zerocost 1 (620)	256	277	288.7±3.7 ($n = 3$)
Mangled Zerocost 2 (620)	256	277	285.0±0.8 ($n = 3$)
Mangled Zerocost 3 (620)	256	279	286.7±0.9 ($n = 3$)
Original Zerocost (620)	256	279	287.2±2.4 ($n = 10$)

Table 4.2.2: Total coverages of $[f, h, \textit{fifo}]$, $[f, h, \textit{lifo}]$ and $[f, h, \langle d \rangle, \textit{ro}]$ (with three seeds). Each row represents the original set of domains or its three action-mangled variants. The effect of action ordering is small enough for $[f, h, \langle d \rangle, \textit{ro}]$ to constantly perform better than the traditional tiebreaking methods. Note: We used the randomized version of $\langle d \rangle$ in this experiment.

Chapter 5

New Perspective: Optimal Search as a Sequence of Satisficing Searches

So far, we have shown that by carefully analyzing search within an f-cost plateau, we were able to develop an effective *knowledge-free*, depth-based tie-breaking method which can significantly improve search performance on domains with 0-cost actions (Table 4.2.1). We now propose a more general framework which underscores the importance of tie-breaking in A^*. *Cost-optimal search can be seen as a series of satisficing searches on each plateau*. In this framework, the problem of tie-breaking can be reduced to a satisficing search.

While A^* requires the first sorting criterion f to use an admissible heuristic in order to find an optimal solution, there are no requirements on the second or later sorting criterion. This means that the search within the same f plateau can be an arbitrary satisficing search[1] without any cost minimization requirement. For

[1]This refers to any algorithm which seeks a satisficing solution, as opposed to the "satisficing" track setting in IPC which also seeks to minimize the plan cost with anytime algorithms

example, if we ignore the first sorting criterion in the standard admissible strategy $[f, h, \textit{fifo}]$, we have $[h, \textit{fifo}]$, which is exactly the same configuration as a Greedy Best First Search (GBFS) using *fifo* default tie-breaking. This means that within a particular f-cost plateau, $[f, h, \textit{fifo}]$ is performing a satisficing GBFS. As another example, the reason for the poor performance of $[f, \textit{fifo}]$ is clearly that it is running $[\textit{fifo}]$, an uninformed satisficing breadth-first search in the plateau.

From this perspective, we can reinterpret A^* as in Algorithm 3: A^* expands the nodes in best-first order of f value. When the heuristic function is admissible and consistent, the f values of the nodes expanded by A^* never decreases during the search process. Thus, the entire process of A^* can be considered as a series of search episodes on each *plateau* (f). The search on each plateau terminates when the plateau is proven to contain no goal nodes (UNSAT), or when a goal is found (SAT). When the plateau is UNSAT, then the search continues to the plateau with the next smallest f value. Figure 5.0.1 also illustrates this framework.

Algorithm 3 Reinterpretation of A^* as iterations of satisficing search on plateaus

```
loop
    Search plateau (f) for any goal state, using satisficing search algorithm
    if plateau (f) contains some goal (Plateau is SAT) then
        return solution
    else
        Increase f
```

This is somewhat similar to the standard approach to model-based planning using SAT/IP/CP solvers (Kautz & Selman, 1992; van den Briel & Kambhampati, 2005), based on an iterative strategy where a planning problem is converted to a corresponding constraint satisfaction problem with a finite horizon t (plan length

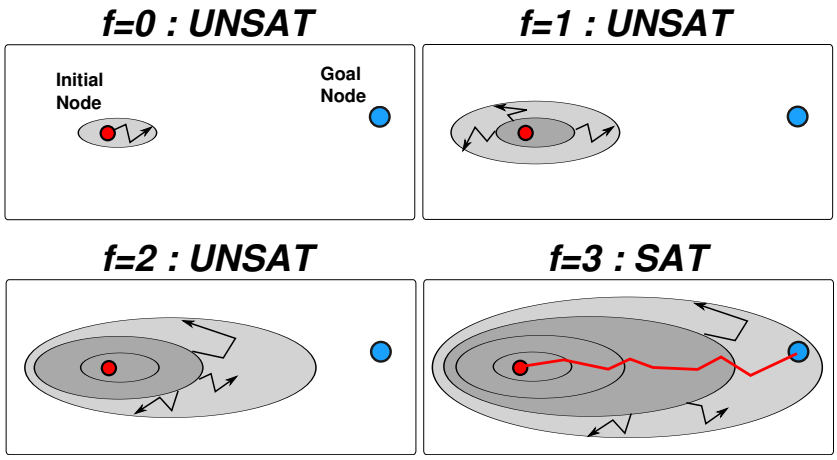

Figure 5.0.1: The concept of A^* as a sequence of satisficing searches.

/ makespan). The search starts from the horizon 0 and tests if the problem is satisfiable. If not, then it increases the horizon, add constraints excluding solutions below t, and retests the same problem with additional constraints for a new horizon $t + 1$.

It is also reminiscent of the behavior of iterative deepening A^* (Korf, 1985), which executes a series of satisficing searches with an f-cost limit which increases on each iteration. However, "A^*-as a sequence of satisficing search" differs from IDA* in that IDA*, in order to achieve linear memory usage, repeats previous work on each iteration. Instead of searching a particular plateau in each iteration, IDA* searches through the union of several plateaus.

The framework of "A^* as a series of satisficing searches" suggests that we can directly apply satisficing search techniques to optimal search using A^*, especially for each f-cost plateau search. In the following two subsections, as well as in the next section, we show that this framework (1) provides a better understanding of depth-diversification (Section 5.1), (2) allows us to prove the completeness of

A^* on infinite graph depending on the tie-breaking methods (Section 5.2), and (3) allows us to improve the performance of A^* on Zerocost domains (Section 5.3).

5.1 Depth Diversification as Satisficing Search

Within this framework, the implementation of depth diversification can be viewed as a variant of the Type-based diversification approach (Xie, Müller, Holte, & Imai, 2014), specifically tailored for Zerocost domains.

Xie et al. proposed *type based buckets*, an implementation of the OPEN list which partitions the nodes into buckets according to some set of key values (*types*). They proposed several types such as $\langle 1 \rangle$, $\langle g \rangle$, $\langle h \rangle$ or $\langle g, h \rangle$. At each type-based expansion, a randomly selected node from a randomly selected single bucket is selected. For example, with type $\langle g, h \rangle$, a node with $g = 5$ and $h = 3$ is put into a bucket $\langle 5, 3 \rangle$. This mechanism diversifies the search so that it tries to expand the nodes with various distances from the initial state and various distances from a goal state.

They then proposed Type-GBFS, which alternates the expansion between normal GBFS with a $[h, \mathit{fifo}]$ sorting criteria and type-based expansion. This alternating framework addresses a weakness of GBFS: GBFS is solely guided by the heuristic function h, and heuristic errors in h can easily misguide GBFS to spend all of its time in the wrong part of the search space – GBFS can become trapped due to heuristic error and cannot recover from the wrong decision until expanding all nodes in that branch. In the worst case, on infinite graphs, GBFS is not complete because it can be misdirected by the heuristic guidance forever (Valenzano & Xie, 2016). In contrast, in Type-GBFS, the alternation with type-based expan-

sion introduces exploratory behavior of nodes with low g and high h, offering the possibility of escaping from heuristic error traps. As a result, Type-GBFS is probabilistically complete on infinite graphs (Valenzano & Xie, 2016).

Type-GBFS was primarily evaluated in the context of satisficing search with no consideration of plan quality, and performance is solely evaluated according to coverage. Thus, Xie et al adopted a unit-cost domain model: All action costs are ignored and replaced with unit costs in their experiments in order to boost coverage (Xie et al., 2014). This is a commonly used technique for satisficing search which is also used in the first iteration of LAMA2011 (Richter et al., 2011).

In our framework of A^* as a sequence of satisficing searches, depth diversification after h tie-breaking ($[f, h, \langle d \rangle]$) can be viewed as the combination of (1) an implicit transformation of all 0-cost edges within a single *plateau* (f, h) to unit-cost edges, and (2) a pure type-based exploration within that plateau (unlike Type-GBFS, which alternates GBFS and type-based buckets).

The notion of *depth* counts the number of 0-cost actions, which does not change the f value and h value, on the path from the entrance to the current plateau, to the current node. Thus, depth-diversification treats the problem of finding an exit from a particular plateau as a unit-cost satisficing search problem – the depth is analogous to a g-value which is calculated with unit costs and is restricted to a particular plateau.

Depth diversification for tie-breaking in admissible A^* has a different purpose and context from Type-GBFS for satisficing search, and differs as follows. First, depth diversification is focused on finding a satisficing plan *within a single plateau* and on solving *domains with 0-cost actions*. Therefore, depth diversification is applied after the sorting by h. In contrast, type buckets are global — type buckets

have no preceding sorting criteria, and all open nodes are stored in these buckets. Type-GBFS then alternates type buckets and sorting by h, not applying them in a cascade manner.

Nevertheless, the close relationship between depth diversification for admissible A^* and Type-GBFS for satisficing search is important. It shows that if we apply our framework of "A^* as a series of satisficing searches", we can directly use ideas which have been previously proposed for satisficing search within each f-cost plateau search.

5.2 Completeness of A^* on an Infinite Graph

Similarly, we can use this framework for analyzing the completeness of A^* on infinite graphs with respect to various tie-breaking criteria. First, A^* is complete on finite graphs regardless of the tie-breaking strategy (Hart et al., 1968). However, if the graph is infinite, the completeness of the algorithm depends on tie-breaking. We consider several cases depending on which plateau is infinite. We only need to consider plateaus for $f \leq f^*$ because A^* does not expand the nodes in *plateau* (f) for $f > f^*$.

Definition 1. *A graph is* infinite *when the number of nodes in the graph has no upper bound.*

Proposition 1. *If any plateau* (f) *for* $f < f^*$ *is infinite, then* A^* *does not terminate.*

Proof. Algorithm 3 requires proving the UNSAT-isfiability (that there is no solution) of all non-final plateaus, *plateau* (f) $(\forall f < f^*)$, so if any of them are infinite, A^* does not terminate. □

The remaining cases assume that the following two conditions hold: *plateau* (f) is finite $\forall f < f^*$, and *plateau* (f^*) is infinite. Under this assumption, the completeness of A^* using tie-breaking $[f, \text{criterion}_2, \ldots \text{criterion}_k]$ depends on the completeness of the satisficing search algorithm corresponding to $[\text{criterion}_2, \ldots \text{criterion}_k]$ on *plateau* (f^*). For the standard tie-breaking criteria, we can apply previously known results.

Theorem 2 (Valenzano and Xie (2016))**.** *ε-greedy node selection (Valenzano, Schaeffer, Sturtevant, & Xie, 2014) is* probabilistically complete *on an infinite graph, i.e., the probability of finding a solution approaches to 1 when the number of expansion t approaches ∞.*

Corollary 1. *A^* with Random Order tiebreaking $[f, ro]$ is probabilistically complete on an infinite graph when plateau (f) is finite for all $f < f^*$.*

Proof. $[f, ro]$ is an instance of A^* using $[ro]$ as a satisficing algorithm for plateau-search. Since $[ro]$ is a special case of ε-greedy node selection with $\varepsilon = 1$, $[ro]$ is also probabilistically complete on an infinite *plateau* (f^*). □

Breadth-first search is complete when the graph has a finite branch factor below the solution depth. Since FIFO tiebreaking $[f, fifo]$ applies breadth-first search to *plateau* (f^*), it follows that

Proposition 2. *A^* with FIFO tiebreaking $[f, fifo]$ is complete on an infinite graph when plateau (f) is finite $\forall f < f^*$ and the maximum outdegree of the nodes is finite in plateau (f^*) below the solution depth.*

LIFO tie-breaking behaves equivalently to a depth-first search with duplicate detection (DFS-dup) on *plateau* (f^*). Assuming a fixed successor ordering, we get the following:

Proposition 3. *A^* with LIFO tiebreaking $[f, \mathit{lifo}]$ is complete on an infinite graph iff plateau (f) is finite for all $f \leq f^*$.*

Proof. If *plateau* (f^*) is infinite, then either the maximum depth or the maximum outdegree of the nodes is infinite (has no upper bound). If the maximum depth has no upper bound, DFS-dup requires an arbitrary longer runtime before the first backtracking unless the solution is found before it. If the maximum outdegree has no upper bound, there is a successor ordering which forces DFS-dup to search all subtrees that do not contain solutions, and the size of the subtrees has no upper bound. If both the maximum depth and the maximum outdegree are finite, then *plateau* (f^*) is finite and DFS-dup is complete. Combined with Proposition 1, LIFO tie-breaking requires a finite *plateau* (f) for all $f \leq f^*$. □

Finally, we show the completeness of our iteration-based depth diversification in Algorithm 2.

Theorem 3. *A^* with Depth Diversification $[f, \langle d \rangle, *]$ is complete when plateau (f) is finite $\forall f < f^*$ and the maximum outdegree of the nodes is finite in plateau (f^*) for depths below the solution depth.*

Proof. Any solution must have a finite depth d^* on *plateau* (f^*). On every iteration of the pop method of Algorithm 2 from the largest depth D to the depth 0, each depth is expanded once. Since the maximum outdegree is finite, every node with depth $d \leq d^*$ will be expanded in a finite number of iterations. □

5.3 Tie-Breaking Using Distance-to-Go Estimates

In the previous section, we proposed a framework which views cost-optimal A^* search as a series of satisficing searches on each f-cost plateau, and argued that the problem of tie-breaking can be reduced to a satisficing search. We showed that the depth diversification tie-breaking criterion, which is highly effective on Zero-cost domains, is in fact a case where a previously studied technique for satisficing search (type-based exploration) turns out to be highly effective when applied to tie-breaking. In this section, we push this insight further and propose another approach to improving the search performance in plateaus produced by Zerocost domains – using inadmissible *distance-to-go* estimates (heuristics) as a tie-breaking criterion within an admissible A^* search.

Distance-to-go estimates are a class of heuristics which treat all actions as if they have unit cost. Even when 0-cost actions are present, these estimates can predict the number of operations required to reach a goal. In general, the estimates are inadmissible (unless the estimates are guaranteed to underestimate the number of required actions and all actions in the original domain have unit cost). Previous work on distance-to-go-heuristics has focused on their use for satisficing planning.

A^*_ϵ (Pearl & Kim, 1982) is one of the earliest algorithms that combines distance-to-go estimates with the cost estimates. It is a bounded-suboptimal search which expands nodes from the *focal* list, the set of nodes with $f(n) \leq w \cdot f_{min}$ where weight w serves as a suboptimality bound, similar to weighted A^*, and f_{min} is the minimum f value in the OPEN list. While f is based on an admissible heuristic function, the nodes in the focal list are expanded in increasing order of an inadmissible distance-to-go estimate $\hat{h}$. Since the search does not follow the best-first order according to f, it is not admissible, and is instead w-admissible. One ex-

ception is the case of $w = 1$ where the focal list is equivalent to the f plateau and the expansion order in the focal list corresponds to the tie-breaking on plateaus. In our notation, this algorithm can be written[2] as a BFS with the following sorting criteria:

$$[\lceil \frac{f}{w \cdot f_{min}} \rceil, \hat{h}, *]$$

This notation is derived from the fact that the focal list "blur"s f up to $w \cdot f_{min}$. For example, when $w = 2, f_{min} = 5$ and $f(n) = 5, 9, 11$, then $\lceil \frac{f}{w \cdot f_{min}} \rceil = 1, 1, 2$ respectively.

Continuing this line of work, Thayer and Ruml (2009, 2011) evaluated various distance-to-go configurations of Weighted A^*, Dynamically Weighted A^* (Pohl, 1973) and A^*_ϵ, where some configurations use distance-to-go as part of tie-breaking. This work focused on bounded-suboptimal search rather than cost-optimal search. Cushing et al. (2010) pointed out the danger of relying on cost estimates in a satisficing search by investigating "ε-cost traps" and other pitfalls caused by cost estimators for search guidance. Finally, the FD/LAMA2011 satisficing planner incorporates distance-to-go estimates in its iterated search framework (Richter et al., 2011). The first iteration of LAMA uses distance-to-go estimates combined with various satisficing search enhancements.

Benton et al. (2010) proposed an inadmissible technique for temporal planning where short actions are hidden behind long actions and do not increase makespan. Such actions cause "g-value plateaus", which are similar to the large plateaus caused by 0-cost actions in sequential planning. They implemented an inadmissible heuristic function combined with distance-to-go estimates as an extension of Temporal Fast Downward (Eyerich, Mattmüller, & Röger, 2009).

[2] However, an actual implementation may differ due to dynamic updates to f_{min}.

5.4 Embedding Distance-to-Go in Admissible Search

Although previous work on distance-to-go estimates assume a satisficing context, we show that distance-to-go estimates can be useful for cost-optimal search. Since the admissibility of the sorting strategy and the optimality of the solution are not affected by the second or later levels of sorting criteria, it is possible to use an inadmissible distance-to-go estimate in these subsequent sorting criteria without sacrificing the optimality of the solution found. This means inadmissible heuristics can be used for tie-breaking.

Let h be an admissible heuristic function, and $\hat{h}$ be a distance-to-go variation of h, i.e., $\hat{h}$ uses essentially the same algorithm as h, except that while h uses the actual action costs for the problem domain, $\hat{h}$ replaces all action costs with 1. Since h is admissible, multi-heuristic sorting strategies such as $[g + h, h, \hat{h}]$ or $[g + h, \hat{h}]$ are admissible.

Moreover, we can even use a multi-heuristic strategy which uses an inadmissible heuristic for tie-breaking which is unrelated to the primary, admissible heuristic h. For example, $[g + h^{\text{LMcut}}, \hat{h}^{\text{FF}}]$ is an admissible sorting strategy because the first sorting criterion $f = g + h^{\text{LMcut}}$ uses an admissible LMcut heuristic. Its second sorting criterion, the distance-to-go FF heuristic (Hoffmann & Nebel, 2001), does not affect the admissibility of this entire sorting strategy.

A potential problem with sorting strategies which use multiple heuristics is the cost of computing additional heuristic estimates. For example, $[g + h^{\text{LMcut}}, \hat{h}^{\text{FF}}]$ requires more time to evaluate each node compared to a standard tie-breaking strategy such as $[g + h^{\text{LMcut}}, h^{\text{LMcut}}]$ because computing the $\hat{h}^{\text{FF}}$ heuristic incurs significant overhead per node while the results of h^{LMcut} can be reused by a caching mechanism. When the inadmissible heuristic for tie-breaking is $\hat{h}$, i.e. a distance-

to-go (unit cost) variant of the primary, admissible heuristic h, it may be possible to reduce this overhead to some extent by implementing h and $\hat{h}$ so that they share some of the computation – this is a direction for future work.

5.4.1 Distance-to-Go Estimates with Default Tie-Breaking

Tie-breaking using distance-to-go estimates can still leave a set of nodes which are equivalent up to the distance-to-go criterion (multiple nodes can have the same f, h, and $\hat{h}$ values), so additional level(s) of tie-breaking are necessary in order to select a single node. By adding a standard default criterion such as *fifo*, *lifo*, *ro*, we obtain a sorting strategy that imposes a total order. For example, $[f^{\text{LMcut}}, \hat{h}^{\text{FF}}, \textit{fifo}]$ applies *fifo* after the distance-to-go estimate $\hat{h}^{\text{FF}}$.

5.4.2 Distance-to-Go Estimates with Depth Diversification

Furthermore, it is possible to combine depth diversity based tie-breaking with distance-to-go estimates by applying the depth-diversity criterion after the distance-to-go estimate. For example, $[f^{\text{LMcut}}, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{fifo}]$ applies depth diversification criterion after the $\hat{h}^{\text{FF}}$ distance-to-go estimate. As we shall see below, a sorting strategy which performs tie-breaking using both distance-to-go estimates and depth diversity results in the best performance overall.

5.5 Evaluation of Distance-to-Go Estimates as Tie-Breaking Criteria for Admissible Search

We tested various admissible sorting strategies on IPC domains and Zerocost domains. The configurations are listed in Table 5.5.1. In all configurations, the first sorting criterion is the $f = g + h$ value where h is an admissible heuristic (either LMcut or M&S) using the actual action-cost based cost calculation. As the second (and third) criteria, we used $\hat{h}$, the distance-to-go version tested of the original heuristic function h, as well as a distance-to-go variation of FF heuristic ($\hat{h}^{\text{FF}}$). We also added configurations with the depth metric within *plateau* $\left(f, \hat{h}^{\text{FF}}\right)$. A summary of the results is shown in Table 5.5.2. Detailed per-domain results are shown in Tables 9.6.1 - 9.6.4.

(1) $[h+g, \quad h, \quad *]$	(2) $[h+g, \quad h, \quad \hat{h}, \quad *]$	(3) $[h+g, \quad \hat{h}, \quad *]$
(4) $[h+g, \quad \hat{h}^{\text{FF}}, \quad *]$	(5) $[h+g, \quad h, \langle d \rangle, \quad *]$	(6) $[h+g, \quad \hat{h}^{\text{FF}}, \langle d \rangle, \quad *]$

Table 5.5.1: Configurations compared in this section. h is either LMcut or M&S.

5.5.1 Evaluation on Zerocost Domains

In Zerocost domains, we see that $\hat{h}$ tie-breaking outperforms h tie-breaking for both LMcut (e.g. $256 \rightarrow 295$ with *fifo*) and M&S (e.g. $280 \rightarrow 308$ with *fifo*). Also, combining h and $\hat{h}$ can further improve performance when the heuristic is LMcut (e.g. $295 \rightarrow 305$ with *fifo*). The results of combining h and $\hat{h}$ were comparable to $\hat{h}$ when the main heuristic function h is M&S. Yet more surprisingly, using $\hat{h}^{\text{FF}}$ further improved the performance for both LMcut (e.g. $[f, h, \hat{h}, \textit{fifo}] : 305 \rightarrow$

Sorting Criteria	Zerocost(620) $h = $ LMcut	Zerocost(620) $h = $ M&S	IPC (1104) $h = $ LMcut	IPC (1104) $h = $ M&S
Baselines				
$[f, h, \textit{fifo}]$	256	280	558	491
$[f, h, \textit{lifo}]$	279	301	565	**496**
$[f, h, \textit{ro}]$	261.9 $\pm$ 1.4	287.7 $\pm$ 3.2	558.9 $\pm$ 2.1	489.4 $\pm$ 1.0
$[f, h, \langle d \rangle, \textit{fifo}]$	284	302	571	487
$[f, h, \langle d \rangle, \textit{lifo}]$	264	288	**575**	487
$[f, h, \langle d \rangle, \textit{ro}]$	288.1 $\pm$ 1.6	308.1 $\pm$ 2.1	571.4 $\pm$ 1.7	485.6 $\pm$ 1.5
Distance-to-Go				
$[f, \hat{h}, \textit{fifo}]$	295	308	534	477
$[f, \hat{h}, \textit{lifo}]$	303	305	534	475
$[f, \hat{h}, \textit{ro}]$	301.0	307.3 $\pm$ 1.5	534 $\pm$ 2.1	470.4 $\pm$ 0.9
$[f, h, \hat{h}, \textit{fifo}]$	305	307	536	476
$[f, h, \hat{h}, \textit{lifo}]$	309	306	535	475
$[f, h, \hat{h}, \textit{ro}]$	305.9 $\pm$ 2.1	307.8 $\pm$ 1.4	534.7 $\pm$ 1.5	470.9 $\pm$ 0.9
$[f, \hat{h}^{\text{FF}}, \textit{fifo}]$	337	336	564	458
$[f, \hat{h}^{\text{FF}}, \textit{lifo}]$	340	331	562	457
$[f, \hat{h}^{\text{FF}}, \textit{ro}]$	341 $\pm$ 2.2	**337.9 $\pm$ 2.1**	563.7 $\pm$ 1.4	457 $\pm$ 1.3
Distance + Depth				
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{fifo}]$	*340* ($>$ 337)	*337* ($>$ 336)	563	457
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{lifo}]$	*342* ($>$ 340)	*333* ($>$ 331)	560	457
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{ro}]$	**344.3 $\pm$ 1.8**	**337.6 $\pm$ 1.3**	561.9 $\pm$ 1.4	456.8 $\pm$ 1.2

Table 5.5.2: Summary Results: Coverage comparison (the number of instances solved in 5min, 4GB) between several sorting strategies. For comparison, we also include the results of configurations evaluated in the previous sections.

$[f, \hat{h}^{\text{FF}}, \textit{fifo}] : 337$) and M&S (e.g. $[f, h, \hat{h}, \textit{fifo}] : 307 \rightarrow [f, \hat{h}^{\text{FF}}, \textit{fifo}] : 336$). Thus, when the depth diversity criterion is not used, the best configurations are those which use $\hat{h}^{\text{FF}}$.

The reason for the good performance of $[f^{\text{LMcut}}, \hat{h}^{\text{FF}}, *]$ is not surprising: $\hat{h}^{\text{FF}}$ is by itself known to be a powerful inadmissible heuristic function for satisficing GBFS, and if we ignore the first sorting criterion, $[f^{\text{LMcut}}, \hat{h}^{\text{FF}}, *]$ is a GBFS with $[\hat{h}^{\text{FF}}, *]$.

Adding the depth diversity criterion further improves the performance of the $\hat{h}^{\text{FF}}$-based strategies, although the impact was small. The coverage increased in both $h = h^{\text{LMcut}}$ (*fifo*: $337 \rightarrow 340$, *lifo*: $340 \rightarrow 342$, *ro*: $341 \rightarrow 344.3$) and $h = h^{\text{M\&S}}$ (*fifo*: $336 \rightarrow 337$, *lifo*: $331 \rightarrow 333$). When the default tie-breaking was *ro* and the heuristic is M&S, $[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{ro}]$ performed slightly worse than $[f, \hat{h}^{\text{FF}}, \textit{ro}]$, but the difference was very small ($337.9 \rightarrow 337.6$) and $\langle d \rangle$ made the performance slightly more robust (smaller standard deviation: $2.1 \rightarrow 1.3$).

5.5.2 Evaluation on Standard IPC Domains

For the standard IPC benchmark instances, the overhead due to the additional computation of $\hat{h}$ or $\hat{h}^{\text{FF}}$ tends to harm the overall performance. Therefore, the best configuration using LMcut was $[f, h, \langle d \rangle, \textit{lifo}]$ which uses depth and does not impose the cost of additional heuristics, and the best result using M&S was $[f, h, \textit{lifo}]$ which imposes no overhead including the depth.

If we look further into the detail, we observed the following: In Cybersec, distance-to-go variants (e.g. $[f^{\text{LMcut}}, \hat{h}^{\text{FF}}, \textit{lifo}]$:5) improve upon the standard strategy (e.g. $[f^{\text{LMcut}}, h^{\text{LMcut}}, \textit{lifo}]$:3), but does not improve upon depth (e.g. $[f, h, \langle d \rangle, \textit{lifo}]$: 12). When $h = h^{\text{M\&S}}$, all coverages are zero. Overheads by $\hat{h}^{\text{FF}}$ also slightly de-

grade the performance in Openstacks (e.g. $[f^{\text{LMcut}}, h^{\text{LMcut}}, \mathit{lifo}]$:18, $[f^{\text{LMcut}}, \hat{h}^{\text{FF}}, \mathit{lifo}]$:17, $[f^{\text{LMcut}}, h^{\text{LMcut}}, \langle d \rangle, \mathit{lifo}]$: 18; Also, $[f^{\text{M\&S}}, h^{\text{M\&S}}, \mathit{lifo}]$:19, $[f^{\text{M\&S}}, \hat{h}^{\text{FF}}, \mathit{lifo}]$:18, $[f^{\text{M\&S}}, h^{\text{M\&S}}, \langle d \rangle, \mathit{lifo}]$: 19). Thus, in these two domains, although there are some improvements in search efficiency due to the guidance by $\hat{h}^{\text{FF}}$ or $\hat{h}$, the runtime overhead of computing the distance-to-go heuristics outweighed the benefit.

In the domains with only positive cost actions (all IPC domains except Openstacks and Cybersec), $\hat{h}$ or $\hat{h}^{\text{FF}}$ only harm the overall performance due to the overhead. When the primary heuristics is LMcut, we do not observe a significant difference between single-heuristics strategies except for the fractional difference in the configurations using *ro*. When the primary heuristic is M&S, $[f^{\text{M\&S}}, h^{\text{M\&S}}, \mathit{lifo}]$ performs slightly better than other default tie-breaking strategies; It also outperforms the depth-based variants as we already discussed in Section 4.2.

5.5.3 Summary of the Evaluation

Table 5.5.3 summarizes the overall conclusions of our performance evaluations. We conclude that although the performance gain by depth diversification and distance-to-go heuristics depend on the domain characteristics, they provide a promising overall performance enhancement.

5.5.4 Simple Dynamic Configuration for Overall Performance

In practice, the performance degradation when using multi-heuristic strategy in domains with only positive cost actions does not pose a problem. We can easily avoid the overhead incurred by the distance-to-go heuristics in those domains by applying the following simple policy: If there are any 0-cost actions, use a multi-

Primary Heuristics	Zerocost domains	Zerocost IPC (Cybersec, Openstacks)	Positive-cost IPC
LMcut	$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{ro}]$	$[f, h, \langle d \rangle, \textit{lifo}]$	$[f, h, *]$ or $[f, h, \langle d \rangle, *]$ (any *default* tie-breaking)
M&S	$[f, \hat{h}^{\text{FF}}, \textit{ro}]$ or $[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{ro}]$, but the latter has a smaller variance.	$[f, h, \textit{lifo}]$ or $[f, h, \langle d \rangle, *]$ (any *default* tie-breaking)	$[f, h, \textit{lifo}]$

Table 5.5.3: Summary of the performance evaluation: Best tie-breaking strategy for each group of domains and each primary heuristic function.

heuristic strategy; Otherwise, use a single-heuristic strategy.

Since the impact of such a check on the total runtime is negligible, we can extrapolate the result of applying this rule based on the previously obtained results. Coverage results in Table 5.5.4 show the total coverage of Zerocost and IPC benchmark domains. The bottom two rows, labeled as *dynamic configuration*, are the extrapolated results when the switching policy is applied – this dynamic configuration achieves the highest overall coverage.

When the configuration rule is applied to standard IPC instances, the domains with 0-cost actions are Cybersec and Openstacks only. They are solved using a multi-heuristic strategy while other domains are solved in the best performing single-heuristic strategy. In Zerocost instances, all domains are solved using the

multi-heuristic strategy.

Overall, these results also strengthen our claim that one should not necessarily rely upon h-based tie-breaking in some domains, as already discussed in Section 3.2.1. In Zerocost domains, using a distance-to-go version of an inadmissible heuristic function for tie-breaking is more effective. Also, combining the depth metric with such an inadmissible heuristics is also effective.

We only tested this relatively simple dynamic configuration that switches between two strategies based on the presence of 0-cost operators. However, as noted in Section 4.2, domain-specific solvers (as opposed to domain-independent solvers, which are the main focus of this thesis) can benefit from fine-tuning the tiebreaking strategy so that it is most suited to the target domain.

	$h =$ LMcut	$h =$ M&S
Single-heuristic strategies		
$[f, h, \textit{lifo}]$	844	797
$[f, h, \langle d \rangle, \textit{fifo}]$	855	789
$[f, h, \langle d \rangle, \textit{lifo}]$	839	775
$[f, h, \langle d \rangle, \textit{ro}]$	859.5	793.7
Multi-heuristic strategies		
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{fifo}]$	903	794
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{lifo}]$	902	790
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{ro}]$	906.2	794.4
Dynamic Configuration		
If a problem contains zerocost actions:		
Then $[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{ro}]$; Else $[f, h, \langle d \rangle, \textit{lifo}]$	**911.9**	
If a problem contains zerocost actions:		
Then $[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{ro}]$; Else $[f, h, \textit{lifo}]$		**832.3**

Table 5.5.4: Summary Results: Coverage comparison, the total number of instances in IPC and Zerocost domains (1724), solved in 5min, 4GB, with several sorting strategies, plus a dynamic configuration strategy. $[f, h, \textit{fifo}], [f, h, \textit{ro}], [f, \hat{h}, *], [f, h, \hat{h}, *], [f, \hat{h}^{\text{FF}}, *]$ are not shown because they achieve smaller coverage.

Chapter 6

Search Diversification for Satisficing Search Algorithms: Intra-vs-Inter Plateau Diversification

Many search problems in AI are too difficult to solve optimally, and finding even one satisficing solution is challenging. Greedy Best-First Search (GBFS) is a best-first search variant where the expansion priority of node n is based only on a heuristic estimate of the node $h(n)$. GBFS has been shown to be quite useful when it is necessary to find some satisficing solution quickly, and GBFS has been the basis for state-of-the-art domain-independent planners.

Despite the ubiquitous use of GBFS for satisficing search, previous work has shown that GBFS is susceptible to being easily trapped by undetected dead ends and huge search plateaus. On infinite graphs, GBFS is not even complete (Valenzano & Xie, 2016) because it could be misdirected by the heuristic guidance forever. These pathological behaviors are caused by the fact that the search behavior

of GBFS strongly depends on the quality of the heuristic function.

Previous approaches to this problem can be classified into two classes. The first class of methods focuses on an issue arising with inadmissible heuristics, which can incorrectly label nodes which are close to the goal (low h^*, the optimal cost to goal) as unpromising (overestimation: $h > h^*$), causing GBFS to delay expanding them until all other open nodes with smaller h-values have been expanded. Several approaches have been proposed for alleviating this problem, including DBFS (Imai & Kishimoto, 2011), ϵ-GBFS (Valenzano et al., 2014) and Type-GBFS (Xie et al., 2014). These approaches *diversify* the search by occasionally expand nodes which do not have the lowest h-value, and provide an opportunity to expand nodes that are mistakenly overlooked due to heuristic errors.

The second class of methods focuses on a different issue which arises in both admissible and inadmissible heuristics: A node that is far from the goal (high h^*) can be mislabeled as promising (underestimation: $h < h^*$), causing GBFS to have larger plateaus and expand unnecessary nodes. Techniques which address this issue include *plateau escaping* (Coles & Smith, 2007), *local exploration* (Xie, Müller, & Holte, 2014) or *tiebreaking* (Asai & Fukunaga, 2016).

All of these methods share the objective of removing some *bias*, thereby encouraging *exploration* by the search process and adding *diversity* in decision-making process. In this thesis, we use the terms "exploration", "diversity", and "bias removal" interchangeably. Previous work lacked a common framework which unified these various approaches to diversification/exploration/bias removal. Furthermore, as shown later, the current state-of-the-art methods are based on diversification with respect to search depth (distance from the start / goal / plateau entrance), so the bias among the set of nodes with the same search depth is not

removed.

In this chapter, we first show that the above two classes of approaches to diversification are orthogonal and should be combined for better performance. We show that a recently proposed depth-based tie-breaking strategy for A^* (Asai & Fukunaga, 2016) also improves the performance of GBFS by diversifying the depth within each h-plateau. Both depth diversification strategy and Type-GBFS are shown to be instances of a *type*-based diversification strategy (Xie et al., 2014): Depth diversification applies type-based diversification within a plateau, and Type-GBFS applies it between plateaus. We compare their empirical performance and show that their improvements are complementary – They improve the performance in different domains, and a configuration using both methods, achieves the best overall coverage. This effectively shows that *inter*-plateau and *intra*-plateau diversification are two orthogonal usages of diversification, and both modes should be used if possible.

Next, we propose and evaluate a new diversification strategy called IP-diversification which addresses diversity with respect to *breadth*. We evaluate this new diversification strategy both for intra-plateau and inter-plateau exploration. Complementary effects on intra/inter-plateau exploration were similarly observed. In addition, IP-diversification outperforms the Type-based diversification strategy. Finally, we show that by combining several intra/inter plateau exploration strategies, we can improve upon state-of-the-art planners in terms of coverage.

6.1 Background

6.1.1 Exploration Mechanisms

One class of improvements to GBFS seeks to introduce exploration (diversity) to the search process, as exemplified by DBFS (Imai & Kishimoto, 2011), ϵ-GBFS (Valenzano et al., 2014), Type-GBFS (Xie et al., 2014). These algorithms address the problem of GBFS getting stuck due to heuristic errors. GBFS will not expand a node n until it expands all nodes with a lower h-value than n. Thus, search progress can be delayed when a good (low-h^*) node is mistakenly assigned a poor (high) h-value (overestimation), or bad (high-h^*) nodes are assigned promising h-values (low-h, underestimation). These exploration strategies allow the search to escape local minima by relaxing the h-based best-first node expansion order.

KBFS(k) (Felner, Kraus, & Korf, 2003) attempts to address this problem by expanding k nodes at a time. ϵ-GBFS (Valenzano et al., 2014) selects a random node from OPEN with some fixed probability $\epsilon < 1$. This is a randomized and weighted alternating OPEN list using $[h, *]$ and $[ro]$ (no sorting criteria). If $\epsilon = 1/2$, the behavior is similar to a deterministic alternation strategy, $alt([h, *], [ro])$.

While ϵ-GBFS relies on a pure randomization strategy to escape traps and introduce exploration, Type-GBFS (Xie et al., 2014) explicitly seeks to remove bias and diversify the search by categorizing OPEN according to several key values, such as $[g, h]$ for each state. Each node is assigned to a bucket according to its key value. The search then selects a random node in a random bucket, avoiding the cardinality bias among buckets. Since Type-GBFS does not sort the buckets according to the key vector, we use a different notation $\langle \ldots \rangle$, such as $\langle g, h \rangle$ denoting type buckets whose key values are g and h. In the implementation evaluated

by Xie et al. (2014), Type-GBFS alternates the exploitative (standard best-first order) expansion and the exploratory (randomized) expansion. We denote this as $alt([h, *], [\langle g, h\rangle, ro])$.

DBFS (Imai & Kishimoto, 2011) diversifies the search based on g and h values, but with several key differences from the above two algorithms: First, the exploratory selection is not uniformly random, but is subject to a particular distribution function based on h, g, h_{min} and g_{max}. Second, it uses a local search with a bounded number of expansions equal to h, which dynamically balances the exploration and exploitation — it does more GBFS when h is large (far from the goal), and less GBFS near the goal (h is small).

GBFS with Local Exploration (GBFS-LE) introduces a 2-level search architecture which runs GBFS until it detects that no improvements have been made for a while, and then runs a local GBFS (GBFS-LS) or random walk (GBFS-LRW) in order to find an exit to a more promising region of the search space (Xie et al., 2014).

6.1.2 Tiebreaking

For GBFS, to our knowledge, there is currently no well-established tie-breaking policy analogous to h-based tie-breaking for A^*. Presumably, this is because while A^* has access to three cost values (f, g, and h), GBFS is guided solely by the heuristic value h.[1] As a consequence, improvements to GBFS have been primarily achieved by addressing other aspects, such as modifying the evaluation scheme

[1] Tie-breaking based on g is sometimes used, but this is motivated as a means to find higher-quality solutions. To our knowledge, in a satisficing context, tie-breaking strategies for reducing search effort have not been explicitly motivated or evaluated.

(Richter & Westphal, 2010, lazy evaluation), queue alternation (multiple heuristic functions), preferred operators (Hoffmann & Nebel, 2001), and diversification.

6.2 Intra- and Inter-plateau Diversification

Previous work on exploration for GBFS address the problem of heuristic errors by occasionally expanding nodes with high h. Since this type of diversification operates across different search plateaus, we refer to these as *inter-plateau* exploration. However, we propose another type of exploration, which we call *intra-plateau* exploration, which works within a particular plateau. This type of exploration only changes expansion order among the nodes within a plateau. We use this new term rather than tiebreaking in order to emphasize its relationship to plateaus.

Existing inter-plateau exploration can be understood as a diversification applied to h^* plateau. Consider a hypothetical 2-dimensional histogram (Figure 6.2.1) of the number of nodes for each pair h, h^*. If both axes were h^* (i.e., h is a perfect heuristic), all nodes would be on the diagonal line $x = y$. However, in reality, h has errors relative to h^*, as would be shown if we projected the histogram to the x-axis. Since low-h^* nodes may have high-h values, it is sometimes reasonable to expand high-h nodes depending on the distribution defined by the problem characteristics and the heuristic function.

However, the converse can also be true – not only can a single h^*-plateau consists of nodes with different h values, a single h-plateau consists of nodes with different h^* values, as would be shown by projecting the histogram to the y-axis in Figure 6.2.1. This leads to an observation that *in the worst case, a naive algorithm may keep expanding bad (high-h^*) nodes within an h-value plateau*.

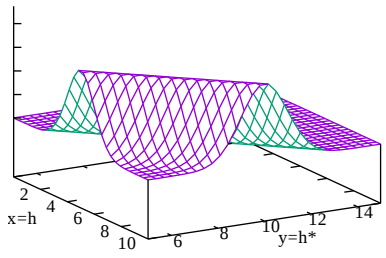

Figure 6.2.1: A conceptual view of the node distribution with regard to h^* and inadmissible h. The peak line on the surface is on $x = y$. Projection to x-axis shows the distribution of h values, while projection to y-axis shows the distribution of h^* values.

More precisely, the node selection algorithm of a diversified GBFS variant can be described as follows:

Definition 2. *An* inter-*plateau diversification strategy for GBFS is a method for selecting the next* h*-value.*

Definition 3. *An* intra-*plateau diversification strategy for GBFS is a method for selecting the next node in the plateau selected by an* inter-*plateau strategy.*

This view cleanly separates the effects of two strategies, providing a firm basis for the observation that their effects are orthogonal and should be combined for the better performance. It is straightforward to see that, given an OPEN list state and an inter-plateau diversification strategy, the next h-plateau to select a node from is independent of intra-plateau strategy. Likewise, given a set of nodes with the same h-value and an intra-plateau (tiebreaking) strategy, the next expanded node is independent of inter-plateau strategy.

Intra-plateau diversification is similar to *local exploration* (Xie et al., 2014; Xie, Müller, & Holte, 2015), but is more restrictive. *Local exploration* is targeted

for uninformative heuristic region (UHR), which includes both plateaus and local minima. In fact, GBFS-LS does not restrict the local exploration by the h-value, thus it may eventually expand a different plateau as the side effect. Similarly, some existing inter-plateau strategies such as ε-greedy GBFS have some intra-plateau side-effects because they may distort the expansion order within a plateau.

6.2.1 Type-Based Diversification

The notion of inter-vs-intra plateau exploration allows us to discuss and compare depth diversification (Asai & Fukunaga, 2016) and Type-GBFS (Xie et al., 2014) within a unified framework – it turns out that they share essentially the same basic idea, while being applied to different contexts (inter-vs-intra plateau, satisficing-vs-optimal search), using different parameters (type systems).

Lelis, Zilles, and Holte (2013) define a general framework for adding exploration to search using "type systems":

Definition 4. *A* Type system *(Lelis et al., 2013) is a function from a node to a vector,* T : *node* $\rightarrow \mathbb{Z}^k, T(n) = \langle t_1(n) \dots t_k(n) \rangle$*, where each function* $t_i(n)$ *returns an integer for each node* n*.*

Xie et al. proposed a node selection technique based on type systems.

Definition 5. Type-Based Node Selection *(Xie et al., 2014) with a type system* $T(\cdot)$ *of* k *types maintains a* k*-dimensional matrix of sets of nodes, where each set* S_v *is associated with a vector* $v = \langle v_1, \dots, v_k \rangle$*. Each node* n *is stored in* $S_{T(n)}$*. For dequeueing, it randomly selects a non-empty set from all sets, and a random node in the set is dequeued.*

The reason for selecting a set at random is to try to allocate the search effort among a diverse set of nodes. Some sets could contain a large number of nodes while others are only scarcely populated. Type-based node selection tries to remove this cardinality bias among buckets. Because type-based node selection has this diversification as an explicit goal and is best understood as a diversification strategy, we call it ***type-based diversification*** in the rest of this thesis.

Type-GBFS (Xie et al., 2014) uses type-based diversification with type system $\langle g, h \rangle$ for inter-plateau exploration. Their inter-plateau exploration is implemented by queue alternation (Röger & Helmert, 2010) between standard Best-First queue and type-based diversification queue.

Depth diversification (Asai & Fukunaga, 2016) originally addressed the issue of zero-cost actions in admissible search with A^*, and the configuration was denoted as $[f, h, \langle d \rangle]$ using the type system notation for a single element d, where $f = g + h$ and d is a number of steps from the current node to the nearest ancestor that has the different h-value. In order to use $\langle d \rangle$ for GBFS, the resulting configuration is $[h, \langle d \rangle]$. This configuration is considered an instance of intra-plateau type-based diversification because it uses type-based diversification $\langle d \rangle$ for diversifying the search within plateaus defined by h.

6.3 Empirical Comparison of Intra- and Inter-Plateau Exploration

Since depth-diversification and Type-GBFS turned out to be instances of the same strategy applied for different purposes (intra/inter-plateau), we use these as exemplars to compare the impact of intra/inter-plateau exploration. In the following

experiments, we empirically show that they achieve complementary performance improvements. This indicates that inter/intra-plateau exploration in fact addresses orthogonal issues of *incorrect* and *insufficient* information, respectively. We then show that intra/inter-plateau exploration can be successfully combined in a single search algorithm.

We compare the performance of the following configurations for Greedy best-first search using the Fast Forward heuristic h^{FF} (Hoffmann & Nebel, 2001) and Causal Graph heuristic h^{CG} (Helmert, 2004).

- **h**: baseline GBFS (eager evaluation).
- **hd**: Depth diversification (Asai & Fukunaga, 2016) – intra-plateau type-based diversification, $[h, \langle d \rangle]$.
- **hD**: Type-GBFS (Xie et al., 2014) – inter-plateau type-based diversification, $alt([h], [\langle g, h \rangle, ro])$,
- **hdD**: A combined configuration of intra- and inter-plateau type-based diversification, $alt([h, \langle d \rangle], [\langle g, h \rangle, ro])$.

Experiments are conducted on a Xeon E5-2666 @ 2.9GHz, HyperThreading and TurboBoost disabled. We used IPC 2011 and 2014 instances with a 4GB memory limit and 5 minutes time limit. Since IPC 2011 and IPC 2014 contain duplicate domains, we removed duplicates from the 2011 set, keeping the 2014 versions. All implementations are based on FastDownward (Helmert, 2006) and unless specified, all configurations use *fifo* default tiebreaking (FastDownward default). Following previous work (Valenzano et al., 2014; Xie et al., 2014), all configurations are evaluated under unit cost transformation because we focused

on the coverage (number of problems solved within resource limit) for purely satisficing search. Each experiment is run 10 times, and the means are shown in Table 6.3.1.

First, intra-plateau exploration **hd** increases coverage for both heuristics h^{CG} ($187 \rightarrow 194.2$) and h^{FF} ($192 \rightarrow 223.9$). This shows that intra-plateau exploration successfully allows GBFS to avoid being trapped in h-value plateaus. Inter-plateau exploration **hD** also increases coverage for both heuristics, confirming the results in (Xie et al., 2014). It is worth mentioning that the performance of **hd** is comparable to **hD**, showing that intra-plateau exploration is no less important than inter-plateau exploration which previous work focused on.

Second, the data shows that the effects of inter/intra-plateau exploration are complementary, as would be expected since they are designed to address orthogonal issues. In most cases, when **hd** improves upon **h** then **hdD** improves upon **hD**, and when **hD** improves upon **h** then **hdD** improves upon **hd**. As a result, for both h^{CG} and h^{FF} heuristics, the **hdD** configuration had higher coverage (h^{CG}:215.8, h^{FF}:223.9) than the hd (h^{CG}:194.2, h^{FF}:208) and hD (h^{CG}:206.1, h^{FF}:207.4) configurations. This shows that combining intra/inter-plateau exploration methods which address orthogonal issues results in better overall performance than either type of exploration by themselves.

Based on these results, we conclude that Inter- and intra-plateau exploration address orthogonal issues and have complementary performance, and combining inter- and intra-plateau exploration can result in better performance than either exploration alone.

These observations imply that *satisficing search can also be reduced to blind search*, similar to what we claimed in the previous chapters (optimal search can

be reduced to satisficing search). This is because the diversification strategies work independently from the heuristic functions (i.e. knowledge-free) and thus are essentially the blind search variants regardless of how it is applied, i.e. either as an intra-plateau or inter-plateau diversification method. As long as we use a single heuristic function h, any search strategy can be described by its behavior on the aforementioned two-dimensional error space, (h, h^*), which is determined by the diversification method = blind search.

Therefore, when the heuristic function being used is fixed, all we need to improve the satisficing search performance is a better blind search algorithm that can be used for inter-plateau and intra-plateau diversification. In the next chapter, we propose a new, randomized blind search algorithm based on Minimum Spanning Tree and fractals.

		h^{CG}				h^{FF}			
		h	hd intra	hD inter	hdD both	h	hd intra	hD inter	hdD both
	total	187	194.2	206.1	**215.8**	192	208	207.4	**223.9**
IPC11 w/o duplicates	elevators	9	8	8.7	**9.7**	19	14	15.9	13.7
	nomystery	7	6	**15.4**	**15.1**	9	7	**16.6**	**17**
	parcprinter	20	20	19.4	18.7	20	20	20	20
	pegsol	20	20	20	20	20	20	20	20
	scanalyzer	20	20	19.9	20	15	15.1	**18**	**18.6**
	sokoban	16	16	**16.9**	**17**	19	19	17.4	17.4
	tidybot	16	**18**	**18.7**	**18.6**	16	16	16	**16.7**
	woodwork	2	2	**2.7**	**7.7**	2	2	**4**	**7.2**
IPC14	barman	0	0	0	0	0	0	**1.5**	**1**
	cavediving	7	7	7	7	7	7	7	7.2
	childsnack	1	**6**	0.1	**1.5**	0	**4**	0	0.3
	citycar	0	0	**7.8**	**4.7**	0	0	**7.2**	**7.1**
	floortile	0	0	**2**	**2**	2	2	2	2.1
	ged	0	0	**9.6**	**9.7**	19	19	14	13.8
	hiking	18	16.9	**19.5**	**19.7**	20	20	19.8	20
	maintenance	16	16	16.1	15.8	11	8	10.7	11.1
	openstacks	0	**3.5**	0	**0.5**	0	**12.6**	0	**7**
	parking	7	**9.7**	1.2	4.1	4	**7.5**	1.4	**5.7**
	tetris	18	17.1	12.4	14.3	1	**5.8**	3.2	**4.9**
	thoughtful	5	5	5	5	8	**9**	**12.7**	**13.1**
	transport	5	3	3.7	4.7	0	0	0	0
	visitall	0	0	0	0	0	0	0	0

Table 6.3.1: Number of solved instances (5 min, 4GB RAM), mean of 10 runs. **h**: baseline GBFS. **hd/hD**: intra/inter-plateau type-based diversification $[h, \langle d \rangle]$ and $alt([h], [\langle g, h \rangle, ro])$ (Type-GBFS), **hdD**: A combined configuration, $alt([h, \langle d \rangle], [\langle g, h \rangle, ro])$. **Bold** indicates that (improvements vs. baseline)> 0.5. Blue indicates that **hdD** improvement correlates with **hd** (intra-plateau) improvement, red indicates that **hdD** improvement correlates with **hD** (inter-plateau) improvement, and orange indicate that both intra/inter-plateau schemes as well as the combined **hdD** scheme improved. Thus, intra- vs. inter-plateau scheme have complementary effects that improve **hdD**.

Chapter 7

Invasion Percolation: Fractal-based Search Diversification

A limitation of type-based diversification based on path distance is that it does not diversify with respect to breadth – nodes with equal estimated distance from goals (h), initial states (g) or plateau entrance (d) are put in a single set. This makes it susceptible to pathological behavior on graphs where some nodes have many more children than others.

Consider a blind search on the directed acyclic graph shown in Figure 7.0.1. The graph consists of two large components, **high-b** and **low-b** branches, and their entries H_1, L_1. The initial search node is I and the goal node is L_4. Both branches have maximum depth D, and the high-b branch has maximum width B. Both B and D are very large. This graph presents a pathological case for all of the previously described methods (*lifo*, *fifo*, *ro* and type-based diversification), depending on successor ordering. *lifo* performs a DFS, and if *lifo* first searches H_1 and the high-b branch due to successor ordering, it must explore the entire high-b

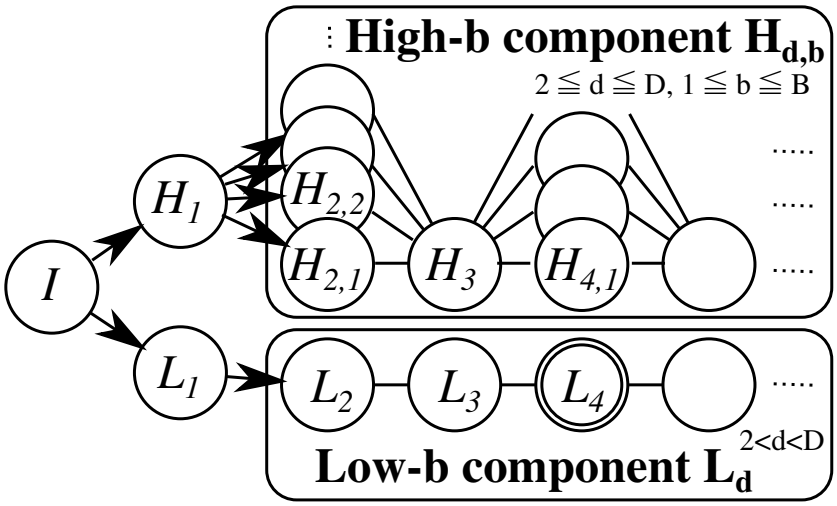

Figure 7.0.1: An example case exhibiting the large bias in the branching factor depending on the subgraph.

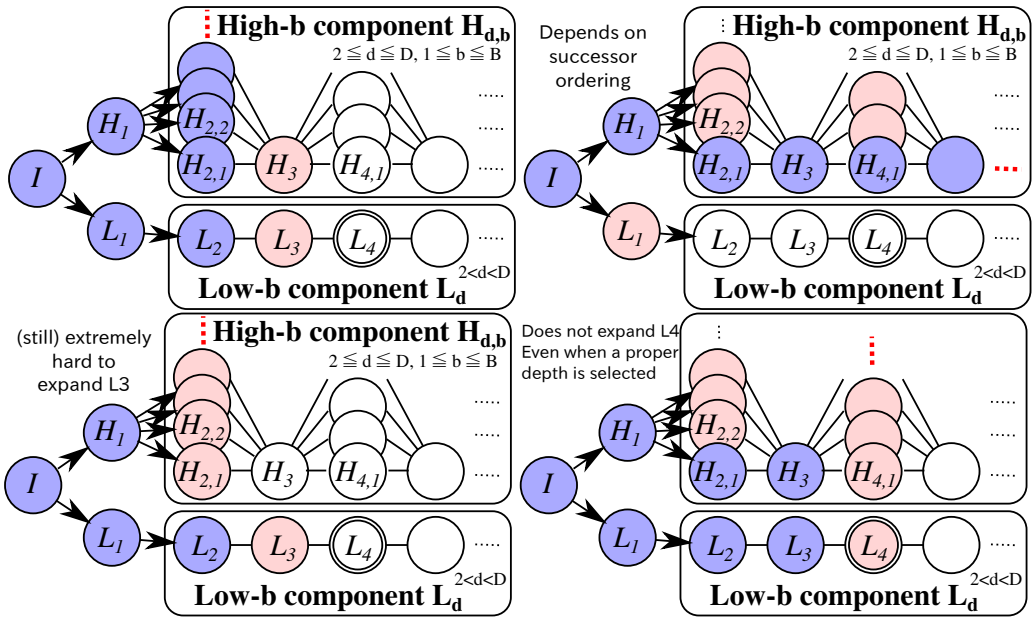

Figure 7.0.2: *fifo*, *lifo*, *ro*, $\langle d \rangle$ all exhibit a pathological behavior due to the large number of successors and the large depth.

branch before expanding L_1 and low-b branch. *fifo* performs Breadth-First Search (BreadthFS), and will therefore suffer from the high branching factor at depth 2 of the high-b branch, getting stuck before reaching L_4. Although randomization can allow *ro* to be better off than the behavior of *fifo*/BreadthFS, the effect is limited: For example, while expanding depth 2, *ro* may occasionally expand depth 3 because it uniformly randomly selects a node from OPEN. However, the probability of expanding nodes at depth 3 is initially only $1/(B+1)$ and continues to be small until most of the nodes at depth 2 are expanded, because OPEN is mostly populated with the nodes from depth 2. Depth-based diversification addresses the depth bias of BreadthFS. However, even though it distributes the effort among various depths, the probability of expanding L_2, L_4 at depths 2 and 4, is only $1/(B+1)$ each, which is very low when B is very large.

We propose *Invasion Percolation-based diversification (IP-diversification)*, a new diversification strategy for satisficing search that addresses this type of bias. IP-diversification combines randomization and Prim's method (Prim, 1957) for Minimum Spanning Tree (MST).

7.1 Invasion Percolation

Invasion Percolation (Wilkinson & Willemsen, 1983) simulates the distribution of fluid slowly invading porous media, e.g., water replacing the air in a porous rock. We focus on a variant called bond IP (BIP), where "bonds" indicate edges in a lattice, and present the graph-based description by Barabási (1996). Given initial node(s) and a graph whose edges are assigned independent random values, BIP iteratively marks the nodes. Once assigned, the random value on each edge

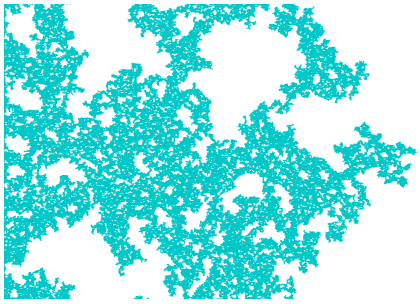

Figure 7.1.1: Invasion Percolation on 2-dimensional lattice. Picture in courtesy of (Monnerot-Dumaine, 2006).

never changes. The initial nodes are marked by default. In each iteration it marks an unmarked node to which the least-value outgoing edge leads. Marked nodes represent the porous sites whose air is replaced by the water (invader). Barabási (1996) showed that this algorithm is equivalent to applying Prim's method for MST (Prim, 1957) on a randomly weighted graph: Prim's method constructs an MST by iteratively adding a neighboring edge with the least edge costs to the existing tree.

Figure 7.1.1 illustrates a 2-D lattice after running BIP for a while. The initial nodes are at the leftmost edge of the rectangular region, i.e. the fluid percolates from the left. The resulting structure has holes of various sizes that the fluid has not invaded, due to the high-valued edges surrounding the neighbors of the holes, which serve as an embankment preventing the water from invading. Since the random value on each edge is fixed, the algorithm does not mark the nodes inside the hole until it marks all nodes with smaller random values in the entire space outside the embankments (Figure 7.1.2). This behavior is critical to forming a fractal structure.

Figure 7.1.2: Embankment effect

7.2 Invasion Percolation for Search Diversification

We adapt the BIP model as an exploration mechanism for best-first search. Previous work on BIP was on physical simulations with relatively small graphs, and to our knowledge, this is the first application of BIP to complex *implicit* graphs.

The actual implementation of BIP is quite simple: A function r_{BIP} returns a randomly selected value for each search edge that caused the node to be evaluated. For each edge, the function should always return the same value once a random value is assigned to that edge. This requires storage whose size is linear in the number of edges that are explored.

For intra-plateau exploration, r_{BIP} is used to break ties in a plateau induced by the primary heuristic function h, i.e. $[h, r_{\text{BIP}}, *]$. Since nodes are sorted in increasing order of the memoized random value attached to each edge, the node expansion order within a plateau follows that of Prim's method. For inter-plateau exploration, we alternate the expansion between standard GBFS and a queue sorted by r_{BIP}: $alt([h], [r_{\text{BIP}}])$, just as in Type-GBFS.

Consider applying BIP to the DAG in Figure 7.0.1. There is a non-negligible

probability that the search finds the solution without expanding high-b branch (Figure 7.2.1): This occurs when the value $v(H_1)$ of H_1 is higher than the value of any of $L_1 \ldots L_4$, whose probability is 1/5 (follows from $\int_0^1 dv(H_1)Pr(\forall i; v(L_i) \leq v(H_1)) = \int_0^1 x^4 dx$). In this case, node H_1 is acting as an embankment, causing nodes in the low-b branch to be expanded. In contrast, the opposite case is very unlikely: L_1 *could* be expanded after expanding all of $H_{d,b}$ for $1 \leq d \leq 4$ and $1 \leq b \leq B$, but the probability of this, $1/(2B+3)$, is very small (assuming large B).

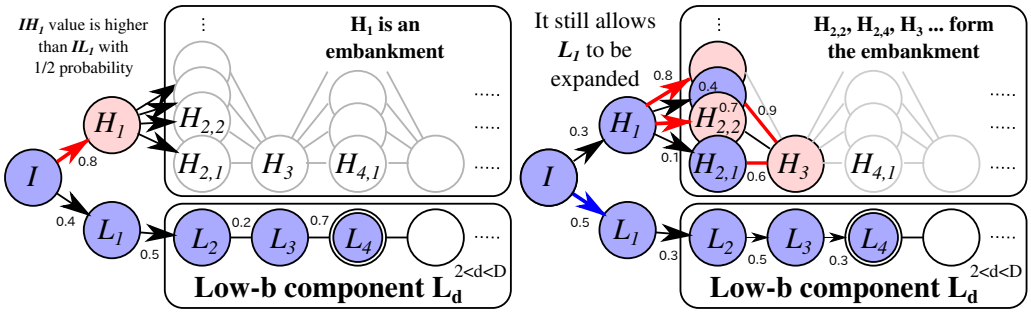

Figure 7.2.1: Examples where BIP successfully prevents the expansion of high-B branch.

Also consider the case when H_1 is expanded with probability $4/5$. Even if this embankment is broken, H_3 could act as another embankment again with probability $1/5$. Moreover, it also avoids expanding a large number of nodes in $H_{2,i}$ whose values are higher than $L_1 \ldots L_4$. $B/5$ of the nodes are not expanded on average because each node is not expanded with the same probability $1/5$.

Thus, at every possible "bottleneck" in the search space that forms an embankment, BIP tends to start looking at the other branches. Since this is affected by the least width of a subgraph rather than the maximum, it is less likely to suffer from the pathological behavior exemplified by Figure 7.0.1.

Node expansion order according to r_{BIP} differs significantly from that of *ro* (pure random selection). *ro* is equivalent to performing a random sort and select the first node, i.e., *ro* essentially assigns a *new* random value to *all* nodes at every single expansion. In contrast, r_{BIP} assigns a value to each edge only once, which develops embankments and allows unexplored "holes" to have longer lifetimes. Consider what would happen if we switch the behavior from r_{BIP} to *ro* starting from the state shown in Figure 7.1.1. Since all nodes are assigned a new random value at each expansion, the embankment nodes are more likely to be expanded, filling the holes more quickly. Thus, running *ro* results in a more solid, denser expansion biased to the left, near the initial nodes.

There is one difference between the assumptions made by BIP/Prim (Barabási, 1996) and classical planning. The search spaces of classical planning are directed while BIP/Prim assumes undirected graphs. Thus, although Prim's method finds the minimum spanning tree on an undirected graph, it may not return the minimum-weight tree on a directed graph. This, however, does not affect the completeness of our search algorithm because it just changes the order of expansion (BIP-based search diversification does not prune any nodes). Adopting algorithms for minimum spanning arborescence for directed graphs (Chu & Liu, 1965; Edmonds, 1967; Tarjan, 1977; Gabow, Galil, Spencer, & Tarjan, 1986) to search diversification is a direction for future work.

7.3 Search Behavior of IP-diversification

We analyze the basic search behavior of IP-diversification by applying a blind search on IPC satisficing instances. We ran four configurations, namely Type-

based diversification with depth d (hd:$[\langle d \rangle]$) and IP-diversification (hb:$[r_{\text{BIP}}]$), as well as BreadthFS (h:$[\mathit{fifo}]$) and random search (ro:$[\mathit{ro}]$). All solvers were given a 3 min/4GB resource limit.

We plotted the depth of the nodes expanded by these algorithms on two representative runs (visitall-sat11-p20, tidybot-sat11-p08) in Figure 7.3.1. As expected, *ro* behaves similarly to BreadthFS/*fifo* (search is biased to the shallow depths) and Depth-diversification shows a flat distribution because it is specifically designed to achieve the fair allocation among depths. Compared to BreadthFS/*fifo* and *ro*, the increase of nodes-per-depth by IP-diversification is much slower, supporting our observation that IP is controlled by the least width in the search graph. Compared to Type-based diversification which shows linear nodes-per-depth, IP still exhibits exponential behavior because IP has no explicit mechanism for balancing the search efforts with regard to depths. However, IP expands the smaller number of nodes in the shallower region. Similar figures were obtained for other domains.

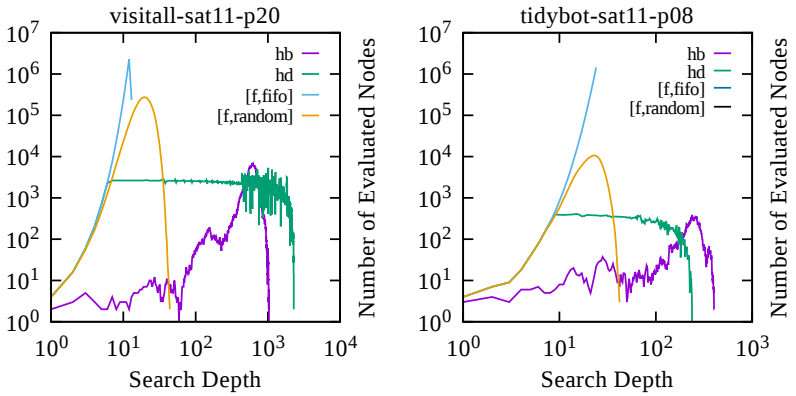

Figure 7.3.1: Distribution of the evaluated nodes per depth.

We also compared their performance on IPC instances. Table 7.3.1 shows that both (hd) and (hb) improves upon blind BreadthFS while not strictly dominat-

ing each other: (hb) shows better performance than (hd) on the Tidybot domain. Comparison between *ro* and hb indicate that the blind performance of IP is better than that of *ro* in tidybot and pegsol.

	h	hb	hd	*ro*
ipc2014 sum	14	15	**22**	15
hiking	2	2	**7**	2
tetris	0	1	**3**	1
ipc2011 sum	30	48	**50.8**	35
pegsol	17	18.5	**19**	17
scanalyzer	4	4	**6**	4
sokoban	3	3	**3.8**	3
tidybot	2	**17.5**	14	6
visitall	0	0	**3**	0

Table 7.3.1: Problems solved with 3 minutes/4GB RAM (average of 10 runs) among 560 instances, using uninformed (blind) diversified search. Best results are in **bold**. We do not show the domains with no differences between configurations.

7.4 Intra- and Inter-Plateau Diversification on a State-of-the-Art Planner

Up to this point, we have evaluated intra/inter-plateau exploration on greedy best-first search in order to cleanly isolate their effect. Next, we evaluate the combined effect of intra/inter-plateau exploration when applied to a state-of-the-art planner, the LAMA2011 configuration in the current version of FastDownward, which in-

corporates a number of search enhancement techniques such as lazy evaluation, multi-heuristic search and preferred operators. In order to focus on coverage, we only run the first iteration (unit-cost GBFS) of LAMA, denoted as

$$alt([h^{\text{FF}}], \textit{pref}\left(h^{\text{FF}}\right), [h^{\text{LC}}], \textit{pref}\left(h^{\text{LC}}\right)),$$

where h^{LC} denotes the landmark-count heuristic and $\textit{pref}(X)$ denotes the preferred operator queue with sorting strategy X.

We apply the methods proposed in this thesis incrementally. We first add a single exploration strategy to LAMA. (d, b) augments $[h]$ with type-based and IP diversification for intra-plateau exploration ($[h, \langle d \rangle]$ and $[h, r_{\text{BIP}}]$), respectively. (D, B) incorporates inter-plateau exploration by adding $\left\langle g, h^{\text{FF}} \right\rangle$ and $[r_{\text{BIP}}]$ to LAMA's alternation queue, respectively. LAMA+D is equivalent to Type-LAMA (Xie et al., 2014). Next, we combine intra/inter-plateau diversification methods: (dD) applies both changes in (d) and (D), and similarly (bB) applies both changes in (b) and (B).

Finally, (db^{2}DB) incorporates all 4 methods into LAMA. Let db denote $alt(\langle d \rangle, r_{\text{BIP}})$, alternation between depth and IP based diversification for intra-plateau exploration, and let DB denote $alt(\left\langle g, h^{\text{FF}} \right\rangle, r_{\text{BIP}})$, alternation between type-based and IP based diversification for inter-plateau exploration. The resulting configuration, LAMA-db^{2}DB, incorporates all of the ideas proposed in this thesis:

$$alt\big([h^{\text{FF}}, db], \textit{pref}\left(h^{\text{FF}}\right), [h^{\text{LC}}, db], \textit{pref}\left(h^{\text{LC}}\right), DB\big).$$

This configuration alternates between type-based and IP diversification in each iteration. It allocates 1/5 of the entire search time to inter-plateau exploration (same as the frequency with which Type-LAMA selects from $\left\langle g, h^{\text{FF}} \right\rangle$), so it spends 1/10

of the time on $[r_{\text{BIP}}]$ and 1/10 of the time on $\langle g, h^{\text{FF}} \rangle$). Adopting more sophisticated approaches for determining exploration frequency (Schulte & Keller, 2014; Nakhost & Müller, 2009) is a direction for future work.

Table 7.4.1 shows the number of solved instances. Each single diversification improved the overall performance of LAMA except LAMA+B. For combinations of two methods (dD and bB), complementary effects by intra-/inter-plateau diversification similar to Table 6.3.1 are observed. Although LAMA+B did not result in improvement, adding B to LAMA+b resulted in larger coverage in LAMA+bB. Finally, bd^{2}BD outperformed all other methods. We observed complementary effects from dD and bB, each addressing different diversity criteria.

7.5 Evaluation of IP-Diversification

Given the performance of blind search, IP-diversification is a good candidate for improving the performance of diversified heuristic search. We compared the performance of (h), the standard GBFS, with the combined Type-based diversification (hdD) from Section 6.3 as well as intra-plateau IP-diversification (hb:$[h, r_{\text{BIP}}]$), inter-plateau IP-diversification (hB:$alt([h], [r_{\text{BIP}}])$), and combined intra/inter-plateau IP diversification (hbB:$alt([h, r_{\text{BIP}}], [r_{\text{BIP}}])$).

Results are shown in Table 7.5.1. IP-diversification, applied to both intra- and inter-plateau exploration, resulted in improvements on both the h^{FF} and h^{CG} heuristics. Complementary effects similar to Table 6.3.1 are observed between hb and hB, and hbB outperforms both hb and hB. This provides additional empirical evidence for the hypothesis that intra/inter-plateau exploration are complementary, and that they can be combined to yield superior performance.

		Planners Based on the Latest FastDownward							
		LAMA	+d	+D	+dD	+b	+B	+bB	+db^{2}DB
	total	293.2	296.5	294.3	295.4	293.3	287.6	297.6	**304.5**
IPC11 w/o duplicates	elevators	20	19.3	19	19.2	20	19.4	19.9	19.6
	nomystery	10	9.9	**17.4**	**16.4**	9.8	10.4	9.7	**16.1**
	parcprinter	20	18.4	19.9	19.7	18.2	19.5	18.3	19.3
	pegsol	20	19	20	20	19.4	20	20	20
	scanalyzer	19	19.3	19.1	19.2	**19.5**	**19.6**	**19.5**	19.2
	sokoban	17	16.9	16.9	16.6	16.4	17	16.9	16.2
	tidybot	16	**17**	15.8	15.8	14.8	15.7	**16.5**	**16.5**
	woodwork	20	20	20	20	20	20	20	20
IPC14	barman	15	13.6	9.5	10.4	12.1	**16**	14.2	14
	cavediving	7	7	7.1	7.1	6.8	6.9	6.7	7
	childsnack	0	**9.3**	0.1	0	0.2	0.3	0.1	0
	citycar	2	1	**5.5**	**4.4**	**4.5**	**4.2**	**4.1**	**4.4**
	floortile	2	2	2.1	2	2	2	2	2
	ged	20	20	20	20	20	20	20	20
	hiking	18.5	18.7	17.5	18.7	**19.1**	17.5	**19.6**	18.8
	maintenance	1	1	**5.5**	**5.6**	1	1	1	**3.6**
	openstacks	20	20	20	20	20	20	20	20
	parking	19.1	**19.8**	16.7	18.7	**19.6**	18.1	18.7	**19.6**
	tetris	9.3	7.1	7.4	7.1	**12.4**	4.7	**15.3**	**14.2**
	thoughtful	14	**14.5**	**15.1**	**15.4**	13.1	**14.5**	12.9	**14.6**
	transport	3.3	**3.8**	2.6	**3.8**	**4.4**	3.7	**3.8**	3.5
	visitall	20	18.9	17.1	15.3	20	17.1	18.4	15.9

Table 7.4.1: Number of solved instances in 5min,4GB RAM. LAMA's sorting strategy is $alt([h^{\text{FF}}], \mathit{pref}\left(h^{\text{FF}}\right), [h^{\text{LC}}], \mathit{pref}\left(h^{\text{LC}}\right))$. For each heuristic $h = h^{\text{FF}}$ and $h = h^{\text{LC}}$ in LAMA, (d,b) augments $[h]$ with type-based and IP diversification for intra-plateau exploration ($[h, \langle d \rangle]$ and $[h, r_{\text{BIP}}]$, respectively). (D,B) applies inter-plateau exploration by adding $\langle g, h^{\text{FF}} \rangle$ and $[r_{\text{BIP}}]$ to LAMA's alternation queue, respectively. D corresponds to Type-LAMA (Xie et al., 2014). (dD) includes both changes in (d) and (D) (similarly for (bB), (b) and (B)). Finally, (db^{2}DB) combines all methods: $alt\left([h^{\text{FF}}, alt(\langle d \rangle, r_{\text{BIP}})], \mathit{pref}\left(h^{\text{FF}}\right), [h^{\text{LC}}, alt(\langle d \rangle, r_{\text{BIP}})], \mathit{pref}\left(h^{\text{LC}}\right), alt(\langle g, h^{\text{FF}} \rangle, r_{\text{BIP}})\right)$. The same highlighting rules as Table 6.3.1 are applied. LAMA+db^{2}DB combines improvements from 4 diversification strategies and achieved the best overall coverage.

		h^{CG}					h^{FF}				
		h	hb intra	hB inter	hbB both	hdD both	h	hb intra	hB inter	hbB both	hdD both
	total	187	187.2	206.8	*208.7*	**215.8**	192	207.8	232.9	**237.7**	223.9
IPC11 w/o duplicates	elevators	9	9.2	**12.6**	**13.3**	**9.7**	19	18.2	18.5	19.4	13.7
	nomystery	7	6.4	5.5	5.6	**15.1**	9	6.6	7.6	6.6	**17**
	parcprinter	20	19.6	13.7	12.4	18.7	20	20	19.9	18.9	20
	pegsol	20	20	19.7	19.8	20	20	20	20	20	20
	scanayzer	20	20	20	20	20	15	16.6	**19.1**	**19.1**	**18.6**
	sokoban	16	15.9	15.8	15.2	**17**	19	18.6	18.5	18.4	17.4
	tidybot	16	**17.3**	**17.5**	**17.5**	**18.6**	16	15	16.4	16.3	**16.7**
	woodworking	2	1.8	**14**	**12.8**	**7.7**	2	1.5	**14.8**	**15.7**	**7.2**
IPC14	barman	0	0	0	0	0	0	0	**7.6**	**6.5**	**1**
	cavediving	7	7.1	7	6.9	7	7	7	7	7	7.2
	childsnack	1	0	0.1	0	**1.5**	0	0	0.1	0	0.3
	citycar	0	0.2	**1.1**	0.4	**4.7**	0	0	**3**	**3.8**	**7.1**
	floortile	0	0	**0.5**	0.2	**2**	2	2	2.1	2	2.1
	ged	0	0	**4.8**	**4.6**	**9.7**	19	19.2	12.8	13	13.8
	hiking	18	15.9	**18.7**	**18.8**	**19.7**	20	17.6	19.9	20	20
	maintainance	16	14.6	14.9	14.1	15.8	11	6.7	10	5.8	11.1
	openstacks	0	0.1	**2.5**	**2.4**	**0.5**	0	**15.7**	**11.7**	**14.5**	**7**
	parking	7	**10.4**	7.6	**10.9**	4.1	4	**5.4**	2.3	**4.8**	**5.7**
	tetris	18	**19.7**	17.6	**19.4**	14.3	1	**8.6**	**7**	**11.1**	**4.9**
	thoughtful	5	4.9	5.2	5.2	5	8	**9.1**	**11.2**	**11**	**13.1**
	transport	5	4.1	**6**	**7.1**	4.7	0	0	0	0	0
	visitall	0	0	**2**	**2.1**	0	0	0	**3.4**	**3.8**	0

Table 7.5.1: Number of solved instances (5 min, 4Gb RAM), mean of 10 runs. **h**: baseline GBFS. **hb/hB**: intra / inter-plateau IP diversification $[h, r_{\mathrm{BIP}}]$ and $alt([h], [r_{\mathrm{BIP}}])$, **hbB**: A combined IP configuration $alt([h, r_{\mathrm{BIP}}], [r_{\mathrm{BIP}}])$, **hdD**: $alt([h, \langle d \rangle], [\langle g, h \rangle, ro])$ (same as hdD from Table 6.3.1) . The same highlighting/-coloring rules as Table 6.3.1 are applied, showing that intra/inter-plateau schemes based on IP are complementary. **bold** shows the improvements by **hdD**. Although **hbB** and **hdD** are comparable overall, per-domain comparison shows **hbB** and **hdD** are complementary.

Overall, hbB performs comparably to hdD. However, note that some domains were improved by Type-based but not by IP (e.g. nomystery, sokoban, child-snack) or vise versa (transport, visitall). These results indicate that Type-based and IP diversification are orthogonal, addressing different diversity criteria (depth vs breadth).

Chapter 8

Related Work

Previous work on escaping search space plateaus has focused on non-admissible search. DBFS (Imai & Kishimoto, 2011) adds stochastic backtracking to Greedy Best First Search (GBFS) to avoid being misdirected by the heuristic function. Type based buckets (Xie et al., 2014) classify plateaus in GBFS according to the $[g, h]$ pair and distributes the effort.[1] Marvin (Coles & Smith, 2007) learns plateau-escaping macros from the Enhanced Hill Climbing phase of the FF planner (Hoffmann & Nebel, 2001). Hoffmann gives a detailed analysis of the structure of the search spaces of satisficing planning (2005, 2011).

Benton et al. (2010) proposed an inadmissible technique for temporal planning where short actions are hidden behind long actions and do not increase makespan. Wilt and Ruml (2011) also analyzes inadmissible distance-to-go estimates. To our knowledge, plateaus have not been previously investigated for cost-optimal search. Admissible and inadmissible search differ significantly in how non-final plateaus (plateaus with $f < f^*$) are treated: Inadmissible search can skip or es-

[1] The relationship between Type-GBFS and our work is discussed in detail in Section 5.1.

cape plateaus whenever possible, while admissible search cannot, unless it is the plateau with $f = f^*$ where the goals can immediately be found.

Some real-time search algorithms like ARA^* (Likhachev, Ferguson, Gordon, Stentz, & Thrun, 2008) are able to prune some states in the final plateau using the knowledge acquired in the previous iterations of suboptimal searches. ARA^* uses a sequence of WA^* ($[g + wh]$) with decreasing weights w, with the final round of iterations being optimal A^* with an uninflated heuristic value (i.e. $w = 1$). When $f = g + wh$ reaches the cost of best path found so far by the previous suboptimal iterations, it can safely terminate the search maintaining the current bounded optimality guarantee w, that is, $w = 1$ in the final iteration. Thus, in an iterated, real-time search setting, this could largely avoid the problem of searching the final plateau if the previous suboptimal searches *happen* to have found the optimal solution already.

In their work on combining multiple inadmissible heuristics in a planner, Röger and Helmert (2010) considered a tie-breaking approach which works as follows: When combining two heuristics h_1 and h_2, h_1 is used as the primary criterion, and h_2 is used to break ties among nodes with the same h_1 — $[h_1, h_2, \textit{fifo}]$. This did not perform well in their work on satisficing planning compared to the approaches based on alternation queues and Pareto-optimal queue selection. Since their focus is on how to combine multiple heuristics, this tie-breaking-based approach was positioned as just one instance of various implementations of OPEN lists. In contrast, this thesis provides a focused, in-depth investigation of various tie-breaking strategies, and shows how tie-breaking enables the efficient search on the plateau created by the earlier levels of sorting criteria.

A^* with lookahead (AL^*) (Stern, Kulberis, Felner, & Holte, 2010) extends A^*

by performing a cost-bounded depth-first *lookahead* from each node as it is generated. Upon the normal expansion of a node n in A^*, lookahead search performs a depth-first search with cost bound $f(n)+k$ rooted at n. As a special case, under the cost bound $k=0$ (AL_0^* in their notation), depth-first lookahead expands only the children with the same f-value. AL^*, or AL_0^* in particular, is similar to $[f, \textit{lifo}]$ in that the lookahead is a depth-first search. However, there are both conceptual and algorithmic differences: First of all, AL_0^* does not specify the intermediate tie-breaking (such as h-based tie-breaking) for its main A^*, and depth-first lookahead does not perform best-first expansion, so the tie-breaking is irrelevant. Thus, the problems and the solutions addressed in these approaches are different. Second, AL^* propagates the maximum and the minimum f values found in the lookahead search, which allows for more pruning.

Another relevant line of work, which is similar in spirit to Zerocost domains, is the Preference Track in the deterministic part of IPC4 (Gerevini, Saetti, & Vallati, 2009). One difference between our Zerocost domains and these domains is that the latter allows a more complex semantics such as multiplication. More recently, Wray et al. (2015) proposed a model called *conditional lexicographic preferences with slack* in the context of planning under uncertainty. Lexicographic preferences allow the problem to have multiple preference criteria evaluated individually. The solution quality is determined by the first preference, breaking ties by the second preference and so on. Slack refers to a constant amount of error from the optimal value. With slack, one can model a situation where the goal is to optimize the first preference, but the difference up to a certain amount is ignored and ties are broken according to the second preference. An example of a planning problem with such lexicographic preferences with slack would be a transportation problem

where the first optimization objective is the amount of fuel usage, allowing a slack up to 5 liters, and the second optimization target is the makespan of the plan. In this case, a plan with 100 liters of fuel usage and a plan with 105 liters of fuel usage are considered equally preferable in the first criterion, and the better plan is the one with a shorter makespan. Since slack allows multiple values (e.g. 100 and 105) to have the same preference, it should introduce larger plateaus. Applying our techniques to problems with slack is an avenue of future work.

We theorized that we can understand the diversification of GBFS with respect to two orthogonal error axes of inter-plateau and intra-plateau errors. Recently, another group of ideas for understanding the GBFS behavior, namely *high water marks* (Wilt & Ruml, 2014) and *benches* (Heusner, Keller, & Helmert, 2017), was proposed. Analyzing the interaction between these ideas and our framework is future work.

While this thesis investigated Bond-IP (the variant of Invasion Percolation which fixes random values to edges), the dual variant which fixes values on nodes is called *Site IP*. Analysis of SIP is a direction for future work as they could have different fractal characteristics (Sheppard, Knackstedt, Pinczewski, & Sahimi, 1999). Valenzano et al. (2014, Section 4.3) evaluated a baseline, knowledge-free heuristic which assigns a random h-value to a node. By itself, this would behave similarly to the ro baseline strategy, if heuristic values are reevaluated for reopened nodes (the default behavior in FastDownward[2]). However, Valenzano et al. disabled node-reopening in all their experimental configurations, which, in effect, fixes the random value for each node and makes them behave similarly to SIP.

[2] `http://hg.fast-downward.org/file/df227b467100/src/search/search_engines/eager_search.cc#l202`

Chapter 9

Conclusion and Future Work

In this thesis, we investigated the cost-optimal search using A^* and the diversification strategies for satisficing search based on GBFS.

Our contributions are as follows: First, we showed that tie-breaking has a significant role in the cost-optimal search using A^* (Chapter 3). We empirically showed that most IPC benchmark instances have large plateaus with regard to f, and most of the search effort is spent in the final plateau with $f = f^*$. We then showed that the commonly used tie-breaking policy based on h value fails to provide guidance in the plateau when problem instances have 0-cost actions and have large plateaus with regard to h. We empirically showed that most of the search effort can be spent in the final plateau with $f = f^*, h = 0$ in some domains, and noted that in such a plateau, the search is controlled solely by the default tie-breaking *fifo*, *lifo* or *ro*. We proposed a new set of benchmark instances for cost-optimal planning, called Zerocost domains, which contain many 0-cost actions. We showed that Zerocost versions of IPC benchmark domains tend to have larger final plateaus with $f = f^*, h = 0$ and pose a new challenge to traditional search

algorithms.

As one approach to improving search performance in Zerocost domains, we proposed a depth metric which measures the distance from the entrance to the plateau (Chapter 4). Using this metric, we described the pathological behaviors of *fifo*, *lifo* and *ro*, proposed a new diversification strategy, theoretically and empirically showed that it avoids the pathological behavior and achieves a better performance.

We then introduced a new interpretation of cost-optimal A^* search as a series of satisficing searches among f-cost plateaus of an increasing order of f (Chapter 5). This perspective led to another approach for effective tie-breaking in Zerocost domains, the use of inadmissible distance-to-go estimates as part of a multi-heuristics tie-breaking strategy. Combination of depth diversification and distance-to-go estimates results in the best overall performance. Although there is an additional cost to compute multiple heuristic values, the overhead can be eliminated by a simple case-based configuration which only uses multiple heuristics when 0-cost actions are present in the problem instance.

We then focused on improving the satisficing search performance, motivated by the result that satisficing search can speed up cost-optimal search (while being also important in its own regard). In such an attempt, we introduced the notion of *Intra-* and *Inter*-plateau exploration in satisficing heuristic search (Chapter 6). While previous work on exploration focused on inter-plateau exploration, we argued that intra-plateau exploration addresses orthogonal issues, and showed that the type-based diversification framework originally developed for inter-plateau diversification could be used to unify intra- and inter-plateau diversification. We then showed empirically that these two modes of diversification have orthogonal,

complementary effects when implemented as diversification strategies for GBFS, and showed that it is possible to combine intra/inter-plateau diversification, resulting in a better performance than either class of strategy alone is used. Furthermore, by proposing the inter/intra-plateau framework for understanding the diversification methods for GBFS, we effectively showed that *satisficing search can be reduced to blind search* because the methods for either of the two modes of diversification are all knowledge-free, blind search algorithms.

Next, we showed that type-based diversification is not sufficient for bias avoidance in graphs where nodes have a largely varying number of neighbors, and proposed IP-diversification, a new breadth-aware diversification strategy which addresses this issue (Chapter 7). We then showed that IP-diversification can be used as either intra- or inter-plateau exploration strategy, i.e., IP is a dual-mode diversification strategy unlike depth-diversification and $\langle g, h \rangle$ type-based diversification, which are specialized for either intra- or inter-plateau exploration. This chapter showcased an example of improving the satisficing search performance in a less ad-hoc manner, i.e., by simply devising a better blind search algorithm. Finally, we showed that incorporating these new ideas (performing both intra / inter-plateau exploration, and both type-based (depth) / IP (breadth) diversification) into FD/LAMA yields state-of-the-art performance on IPC benchmark instances.

Overall, through a series of in-depth theoretical and empirical analyses, we showed that various search algorithms can be understood in a simpler, unified framework. This framework allowed us to transfer the knowledge in satisficing search to cost-optimal search, or to exploit a single blind search method in two modes of diversification. The results obtained in this dissertation lead to two interesting and independent directions for future work.

The first direction is to evaluate the applicability of a much broader variety of satisficing algorithms in the context of optimising search, leveraging our reformulation of A^* as a sequence of satisficing searches. Although we evaluated only one relatively simple, satisficing configuration ($\hat{h}^{\text{FF}}$) in the experiments, many techniques which have previously been developed for satisficing planning can be applied to enhance tie-breaking (plateau-search) in cost-optimal search, including lazy evaluation (Richter & Westphal, 2010), alternating/Pareto open list (Röger & Helmert, 2010), helpful actions (preferred operators) (Hoffmann & Nebel, 2001), random walk local search (Nakhost & Müller, 2009), macro operators (Botea, Enzenberger, Müller, & Schaeffer, 2005; Chrpa, Vallati, & McCluskey, 2015), factored planning (Amir & Engelhardt, 2003; Brafman & Domshlak, 2006; Asai & Fukunaga, 2015) and exploration-based search enhancements (Valenzano et al., 2014; Xie et al., 2014; Valenzano & Xie, 2016).

The second direction for future work is to reformulate the existing satisficing search algorithms into the simpler blind search algorithms, instead of trying to embed them directly as the subroutines for optimal search. Since existing satisficing methods tend to be ad-hoc, integrating them into the standard Best-First Search framework may not be straightforward. However, reformulating existing satisficing search algorithms into blind search algorithms would greatly simplify their applications and analyses. Also, their diversification ability could be sometimes constrained in a single mode of diversification (inter/intra-plateau). By identifying their blind-search reformulations, we could find a way to expand their ability to the other mode that was not addressed in the original algorithm.

Appendix: Detailed Data

This Appendix contains some detailed figures and data which are referenced from the text in the previous sections.

9.1 Detailed Data for Table 3.2.1

Domain	$[f, \mathit{fifo}]$	$[f, \mathit{lifo}]$	$[f, \mathit{ro}]$	$[f, h, \mathit{fifo}]$	$[f, h, \mathit{lifo}]$	$[f, h, \mathit{ro}]$
IPC benchmark (1104)	443	558	448.9 ± 1.3	558	**565**	558.9 ± 2.1
airport(50)	18	26	18 ± 0	**27**	26	25.7 ± 0.5
barman-opt11(20)	0	0	0 ± 0	0	0	0 ± 0
blocks(35)	26	26	26 ± 0	**28**	**28**	**28** ± 0
cybersec(19)	0	3	0 ± 0	2	3	**3.9** ± 1.1
depot(22)	5	5	5 ± 0	6	6	6 ± 0
driverlog(20)	12	13	12 ± 0	13	13	13 ± 0
elevators-opt11(20)	14	15	14 ± 0	15	15	15 ± 0
floortile-opt11(20)	6	6	6 ± 0	6	6	6 ± 0
freecell(80)	8	9	8.7 ± 0.5	9	9	9 ± 0
grid(5)	1	1	1 ± 0	1	1	1 ± 0
gripper(20)	6	6	6 ± 0	6	6	6 ± 0
hanoi(30)	12	12	12 ± 0	12	12	12 ± 0
logistics00(28)	16	18	16 ± 0	**20**	**20**	**20** ± 0
miconic(150)	68	140	68 ± 0	140	140	140 ± 0
mprime(35)	20	**22**	19.9 ± 0.3	21	21	20.9 ± 0.3
mystery(30)	15	16	15 ± 0	16	16	15.2 ± 0.4
nomystery-opt11(20)	12	13	12 ± 0	**14**	**14**	**14** ± 0
openstacks-opt11(20)	11	**18**	11.2 ± 0.4	11	**18**	11.7 ± 0.5
parcprinter-opt11(20)	12	13	12 ± 0	13	13	13 ± 0
parking-opt11(20)	1	1	1 ± 0	1	1	1 ± 0
pathways(30)	4	5	4 ± 0	5	5	5 ± 0
pegsol-opt11(20)	17	17	17 ± 0	17	17	17 ± 0
pipesworld-notankage(50)	13	13	13 ± 0	14	14	14.6 ± 0.5
pipesworld-tankage(50)	7	8	8 ± 0	8	8	8 ± 0
psr-small(50)	48	48	48 ± 0	48	48	48 ± 0
rovers(40)	7	7	7 ± 0	7	7	7 ± 0
scanalyzer-opt11(20)	4	**10**	5.4 ± 0.7	**10**	**10**	**10** ± 0
sokoban-opt11(20)	19	19	19 ± 0	19	19	19 ± 0
storage(30)	14	14	14 ± 0	14	14	14 ± 0
tidybot-opt11(20)	11	12	11 ± 0	12	12	12 ± 0
tpp(30)	6	6	6 ± 0	6	6	6 ± 0
transport-opt11(20)	6	6	6 ± 0	6	6	6 ± 0
visitall-opt11(20)	9	10	9.4 ± 0.5	10	10	10 ± 0
woodworking-opt11(20)	6	9	8.2 ± 0.4	**10**	**10**	**10** ± 0
zenotravel(20)	9	**11**	9 ± 0	**11**	**11**	**11** ± 0

Table 9.1.1: Coverage comparison (the number of instances solved in 5min, 4GB, LMcut heuristics) among the standard baseline tie-breaking algorithms. We highlight the best results when the difference between the maximum and the minimum coverage exceeds 2.

Domain	$[f, \mathit{fifo}]$	$[f, \mathit{lifo}]$	$[f, \mathit{ro}]$	$[f, h, \mathit{fifo}]$	$[f, h, \mathit{lifo}]$	$[f, h, \mathit{ro}]$
IPC benchmark (1104)	460	490	460.9 ± 1.6	491	**496**	489.4 ± 1.0
airport(50)	9	9	9 ± 0	9	9	9 ± 0
barman-opt11(20)	4	4	4 ± 0	4	4	4 ± 0
blocks(35)	21	22	21 ± 0	22	22	22 ± 0
cybersec(19)	0	0	0 ± 0	0	0	0 ± 0
depot(22)	5	6	5 ± 0	6	6	5 ± 0
driverlog(20)	12	12	12 ± 0	12	12	12 ± 0
elevators-opt11(20)	13	13	13 ± 0	13	13	13 ± 0
floortile-opt11(20)	5	6	5 ± 0	6	6	6 ± 0
freecell(80)	15	16	15 ± 0	**17**	**17**	16 ± 0
grid(5)	2	2	2 ± 0	2	2	2 ± 0
gripper(20)	8	**20**	8 ± 0	**20**	**20**	**20** ± 0
hanoi(30)	14	14	14 ± 0	14	14	14 ± 0
logistics00(28)	20	20	20 ± 0	20	20	20 ± 0
miconic(150)	68	**73**	68.3 ± 0.7	**73**	**73**	**73.2** ± 0.4
mprime(35)	23	23	22 ± 0	23	**24**	23.7 ± 0.5
mystery(30)	15	15	15 ± 0	15	16	15 ± 0
nomystery-opt11(20)	17	18	17.8 ± 0.4	18	18	18 ± 0
openstacks-opt11(20)	15	**19**	15.4 ± 0.5	15	**19**	15.4 ± 0.5
parcprinter-opt11(20)	10	10	10 ± 0	10	10	10 ± 0
parking-opt11(20)	1	1	1 ± 0	1	1	1 ± 0
pathways(30)	4	4	4 ± 0	4	4	4 ± 0
pegsol-opt11(20)	17	**19**	17.2 ± 0.4	**19**	**19**	**19** ± 0
pipesworld-notankage(50)	9	9	8.9 ± 0.3	10	10	9.9 ± 0.3
pipesworld-tankage(50)	13	13	13.1 ± 0.3	13	13	13.2 ± 0.4
psr-small(50)	50	50	50 ± 0	50	50	50 ± 0
rovers(40)	6	**8**	6.1 ± 0.3	**8**	**8**	**8** ± 0
scanalyzer-opt11(20)	10	10	10 ± 0	10	10	10 ± 0
sokoban-opt11(20)	20	20	20 ± 0	20	20	20 ± 0
storage(30)	15	15	15 ± 0	15	15	15 ± 0
tidybot-opt11(20)	0	0	0 ± 0	0	0	0 ± 0
tpp(30)	6	6	6 ± 0	7	6	6 ± 0
transport-opt11(20)	7	7	7 ± 0	7	7	7 ± 0
visitall-opt11(20)	9	9	9 ± 0	9	9	9 ± 0
woodworking-opt11(20)	7	7	7 ± 0	7	7	7 ± 0
zenotravel(20)	10	10	10 ± 0	**12**	**12**	**12** ± 0

Table 9.1.2: Coverage comparison (the number of instances solved in 5min, 4GB, M&S heuristics) among the standard baseline tie-breaking algorithms. We highlight the best results when the difference between the maximum and the minimum coverage exceeds 2.

9.2 Detailed Data for Table 4.2.1

	$[f, h, fifo]$	$[f, h, lifo]$	$[f, h, ro]$	$[f, h, \langle d \rangle, fifo]$	$[f, h, \langle d \rangle, lifo]$	$[f, h, \langle d \rangle, ro]$
Zerocost (620)	256	279	261.9 ± 1.4	284	264	**288.1** ± 1.6
airport-fuel(20)	**15**	13	13.8 ± 0.4	14	13	14 ± 0.5
blocks-stack(20)	17	17	17 ± 0	17	17	17 ± 0
depot-fuel(22)	6	6	6 ± 0	6	6	6 ± 0
driverlog-fuel(20)	8	8	8 ± 0	8	8	8 ± 0
elevators-up(20)	7	**13**	7 ± 0	7	9	9.1 ± 0.8
floortile-ink(20)	8	8	8.1 ± 0.3	8	8	8.2 ± 0.4
freecell-move(20)	4	**19**	4.9 ± 0.3	17	10	16.4 ± 0.7
grid-fuel(5)	1	1	1 ± 0	1	1	1 ± 0
gripper-move(20)	7	7	7 ± 0	7	7	7 ± 0
hiking-fuel(20)	9	9	9 ± 0	9	9	9 ± 0
logistics00-fuel(28)	16	16	16 ± 0	16	16	15.3 ± 0.5
miconic-up(30)	16	17	16.6 ± 0.5	19	18	**20.3** ± 0.7
mprime-succumb(35)	15	14	17.1 ± 0.8	**22**	14	20.1 ± 0.3
mystery-feast(20)	7	5	**7.7** ± 0.5	6	5	7.2 ± 0.8
nomystery-fuel(20)	10	10	10 ± 0	10	10	10 ± 0
parking-movecc(20)	0	0	0 ± 0	0	0	0 ± 0
pathways-fuel(30)	5	5	4.3 ± 0.5	5	5	4.1 ± 0.3
pipesnt-pushstart(20)	8	8	8.4 ± 0.5	8	8	**9.8** ± 0.4
pipesworld-pushend(20)	3	4	3.8 ± 0.4	3	3	**4.8** ± 0.4
psr-small-open(20)	19	19	19 ± 0	19	19	19 ± 0
rovers-fuel(40)	8	8	8 ± 0	8	8	8 ± 0
scanalyzer-analyze(20)	9	9	9.1 ± 0.3	9	10	9.2 ± 0.4
sokoban-pushgoal(20)	18	18	18 ± 0	18	18	18 ± 0
storage-lift(20)	4	4	4.1 ± 0.3	5	4	4.2 ± 0.4
tidybot-motion(20)	16	16	16 ± 0	16	16	16 ± 0
tpp-fuel(30)	8	**11**	8 ± 0	**11**	10	**11** ± 0
woodworking-cut(20)	5	7	7 ± 0	8	5	**8.2** ± 0.8
zenotravel-fuel(20)	7	7	7 ± 0	7	7	7 ± 0

Table 9.2.1: Coverage comparison (the number of instances solved in 5min, 4GB, LMcut heuristics) on 620 Zerocost instances. We highlight the best results when the difference between the best and the worst coverages is greater than 2.

	$[f, h, ffifo]$	$[f, h, lifo]$	$[f, h, ro]$	$[f, h, \langle d \rangle, ffifo]$	$[f, h, \langle d \rangle, lifo]$	$[f, h, \langle d \rangle, ro]$
Zerocost (620)	280	301	287.7 ± 3.2	302	288	**308.1** ± 2.1
airport-fuel(20)	5	5	5 ± 0	5	5	5 ± 0
blocks-stack(20)	20	20	20 ± 0	20	20	20 ± 0
depot-fuel(22)	5	5	6 ± 0	6	5	6 ± 0
driverlog-fuel(20)	9	9	9 ± 0	9	9	9 ± 0
elevators-up(20)	8	**14**	8.6 ± 0.5	9	13	11 ± 1
floortile-ink(20)	8	8	8 ± 0	7	7	6.9 ± 0.3
freecell-move(20)	5	17	6.7 ± 0.9	17	15	**17.3** ± 0.5
grid-fuel(5)	2	2	2 ± 0	2	2	2 ± 0
gripper-move(20)	20	20	20 ± 0	20	20	20 ± 0
hiking-fuel(20)	13	13	12.8 ± 0.4	13	12	12.1 ± 0.3
logistics00-fuel(28)	16	16	16 ± 0	16	16	16 ± 0
miconic-up(30)	29	30	30 ± 0	30	30	30 ± 0
mprime-succumb(35)	21	19	19.6 ± 0.7	**25**	15	23.4 ± 0.9
mystery-feast(20)	4	4	5.9 ± 0.3	4	4	**6** ± 0
nomystery-fuel(20)	16	16	16 ± 0	16	16	16 ± 0
parking-movecc(20)	0	0	0 ± 0	0	0	0 ± 0
pathways-fuel(30)	4	4	4 ± 0	4	4	4 ± 0
pipesnt-pushstart(20)	3	3	3.4 ± 0.5	**5**	3	**5** ± 0
pipesworld-pushend(20)	5	**9**	7.7 ± 0.5	5	6	**9** ± 0.9
psr-small-open(20)	19	19	19 ± 0	19	19	19 ± 0
rovers-fuel(40)	8	8	8 ± 0	8	8	8 ± 0
scanalyzer-analyze(20)	11	11	11 ± 0	11	11	11 ± 0
sokoban-pushgoal(20)	19	19	18 ± 0	18	18	18 ± 0
storage-lift(20)	4	4	4 ± 0	4	4	4 ± 0
tidybot-motion(20)	0	0	0 ± 0	0	0	0 ± 0
tpp-fuel(30)	9	10	9.6 ± 0.5	**11**	10	**11** ± 0
woodworking-cut(20)	7	7	8 ± 0.5	8	7	**9** ± 1
zenotravel-fuel(20)	10	9	9.6 ± 0.7	10	9	9.3 ± 1.0

Table 9.2.2: Coverage comparison (the number of instances solved in 5min, 4GB, M&S heuristics) on 620 Zerocost instances. We highlight the best results when the difference between the maximum and the minimum coverage exceeds 2.

IPC benchmark (1104)	$[f, h, \mathit{fifo}]$ 558	$[f, h, \mathit{lifo}]$ 565	$[f, h, \mathit{ro}]$ 558.9 ± 2.1	$[f, h, \langle d \rangle, \mathit{fifo}]$ 571	$[f, h, \langle d \rangle, \mathit{lifo}]$ **575**	$[f, h, \langle d \rangle, \mathit{ro}]$ 571.4 ± 1.7
airport(50)	27	26	25.7 ± 0.5	27	26	25.7 ± 0.5
barman-opt11(20)	0	0	0 ± 0	0	0	0 ± 0
blocks(35)	28	28	28 ± 0	28	28	28 ± 0
cybersec(19)	2	3	3.9 ± 1.1	8	**12**	10 ± 1
depot(22)	6	6	6 ± 0	6	6	6 ± 0
driverlog(20)	13	13	13 ± 0	13	13	13 ± 0
elevators-opt11(20)	15	15	15 ± 0	15	15	15 ± 0
floortile-opt11(20)	6	6	6 ± 0	6	6	6 ± 0
freecell(80)	9	9	9 ± 0	9	9	9 ± 0
grid(5)	1	1	1 ± 0	1	1	1 ± 0
gripper(20)	6	6	6 ± 0	6	6	6 ± 0
hanoi(30)	12	12	12 ± 0	12	12	12 ± 0
logistics00(28)	20	20	20 ± 0	20	20	20 ± 0
miconic(150)	140	140	140 ± 0	140	140	140 ± 0
mprime(35)	21	21	20.9 ± 0.3	21	21	20.9 ± 0.3
mystery(30)	16	16	15.2 ± 0.4	16	16	15.4 ± 0.5
nomystery-opt11(20)	14	14	14 ± 0	14	14	14 ± 0
openstacks-opt11(20)	11	**18**	11.7 ± 0.5	**18**	**18**	**18** ± 0
parcprinter-opt11(20)	13	13	13 ± 0	13	13	13 ± 0
parking-opt11(20)	1	1	1 ± 0	1	1	1 ± 0
pathways(30)	5	5	5 ± 0	5	5	5 ± 0
pegsol-opt11(20)	17	17	17 ± 0	17	17	17 ± 0
pipesworld-notankage(50)	14	14	14.6 ± 0.5	14	15	14.4 ± 0.5
pipesworld-tankage(50)	8	8	8 ± 0	8	8	8 ± 0
psr-small(50)	48	48	48 ± 0	48	48	48 ± 0
rovers(40)	7	7	7 ± 0	7	7	7 ± 0
scanalyzer-opt11(20)	10	10	10 ± 0	10	10	10 ± 0
sokoban-opt11(20)	19	19	19 ± 0	19	19	19 ± 0
storage(30)	14	14	14 ± 0	14	14	14 ± 0
tidybot-opt11(20)	12	12	12 ± 0	12	12	12 ± 0
tpp(30)	6	6	6 ± 0	6	6	6 ± 0
transport-opt11(20)	6	6	6 ± 0	6	6	6 ± 0
visitall-opt11(20)	10	10	10 ± 0	10	10	10 ± 0
woodworking-opt11(20)	10	10	10 ± 0	10	10	10 ± 0
zenotravel(20)	11	11	11 ± 0	11	11	11 ± 0

Table 9.2.3: Coverage comparison (the number of instances solved in 5min, 4GB, LMcut heuristics) on 1104 standard IPC benchmark instances. We highlight the best results when the difference between the maximum and the minimum coverage exceeds 2.

IPC benchmark (1104)	$[f, h, \mathit{fifo}]$ 491	$[f, h, \mathit{lifo}]$ **496**	$[f, h, \mathit{ro}]$ 489.4 ± 1.0	$[f, h, \langle d \rangle, \mathit{fifo}]$ 487	$[f, h, \langle d \rangle, \mathit{lifo}]$ 487	$[f, h, \langle d \rangle, \mathit{ro}]$ 485.6 ± 1.5
airport(50)	9	9	9 ± 0	9	9	9 ± 0
barman-opt11(20)	4	4	4 ± 0	4	4	4 ± 0
blocks(35)	22	22	22 ± 0	22	21	21.9 ± 0.3
cybersec(19)	0	0	0 ± 0	0	0	0 ± 0
depot(22)	6	6	5 ± 0	5	5	5 ± 0
driverlog(20)	12	12	12 ± 0	12	12	12 ± 0
elevators-opt11(20)	13	13	13 ± 0	12	12	12 ± 0
floortile-opt11(20)	6	6	6 ± 0	6	6	6 ± 0
freecell(80)	17	17	16 ± 0	16	16	16 ± 0
grid(5)	2	2	2 ± 0	2	2	2 ± 0
gripper(20)	20	20	20 ± 0	20	20	20 ± 0
hanoi(30)	14	14	14 ± 0	14	14	14 ± 0
logistics00(28)	20	20	20 ± 0	20	20	20 ± 0
miconic(150)	73	73	73.2 ± 0.4	73	73	72.2 ± 0.4
mprime(35)	23	24	23.7 ± 0.5	23	24	23.4 ± 0.5
mystery(30)	15	16	15 ± 0	15	16	15 ± 0
nomystery-opt11(20)	18	18	18 ± 0	18	18	18 ± 0
openstacks-opt11(20)	15	**19**	15.4 ± 0.5	**19**	**19**	**19** ± 0
parcprinter-opt11(20)	10	10	10 ± 0	10	10	10 ± 0
parking-opt11(20)	1	1	1 ± 0	1	1	1 ± 0
pathways(30)	4	4	4 ± 0	4	4	4 ± 0
pegsol-opt11(20)	19	19	19 ± 0	19	19	19 ± 0
pipesworld-notankage(50)	10	10	9.9 ± 0.3	10	9	9.8 ± 0.4
pipesworld-tankage(50)	13	13	13.2 ± 0.4	13	13	13 ± 0
psr-small(50)	50	50	50 ± 0	50	50	50 ± 0
rovers(40)	8	8	8 ± 0	8	8	7.1 ± 0.3
scanalyzer-opt11(20)	10	10	10 ± 0	10	10	10 ± 0
sokoban-opt11(20)	20	20	20 ± 0	19	19	19 ± 0
storage(30)	15	15	15 ± 0	15	15	15 ± 0
tidybot-opt11(20)	0	0	0 ± 0	0	0	0 ± 0
tpp(30)	7	6	6 ± 0	6	6	6 ± 0
transport-opt11(20)	7	7	7 ± 0	6	6	6 ± 0
visitall-opt11(20)	9	9	9 ± 0	9	9	9 ± 0
woodworking-opt11(20)	7	7	7 ± 0	7	7	7 ± 0
zenotravel(20)	**12**	**12**	**12** ± 0	10	10	10.1 ± 0.3

Table 9.2.4: Coverage comparison (the number of instances solved in 5min, 4GB, M&S heuristics) on 1104 standard IPC benchmark instances. We highlight the best results when the difference between the maximum and the minimum coverage exceeds 2.

9.3 Additional Figures for Figure 4.2.1: *lifo* Default Tiebreaking

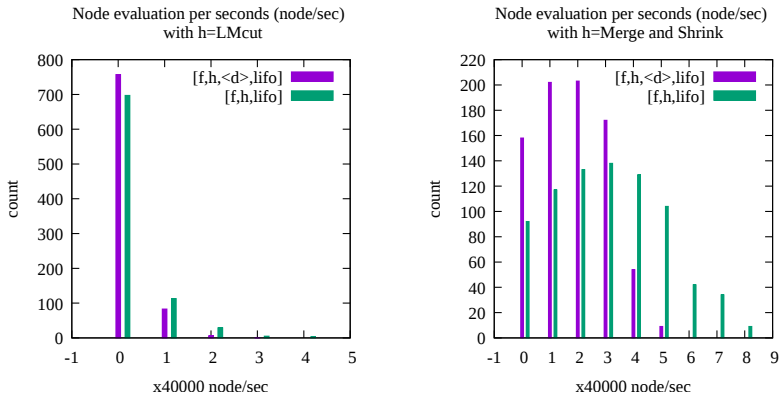

Figure 9.3.1: Histogram comparing the node evaluation ratio (node/sec) between standard tie-breaking ($[f, h, \mathit{lifo}]$) and depth-based tie-breaking ($[f, h, \langle d \rangle, \mathit{lifo}]$) on LMcut and M&S heuristics. On M&S, compared to LMcut, node evaluation rate more often becomes slower when depth is enabled. This is because the node evaluation of M&S is an order of magnitude faster than LMcut, and the overhead of managing depth-based tie-breaking queue becomes significant.

9.4 Additional Figures for Figure 4.2.3: More Histograms for the Size of Final Plateaus

These includes 12 additional histograms for the size of final plateaus on more variety of domains and instances.

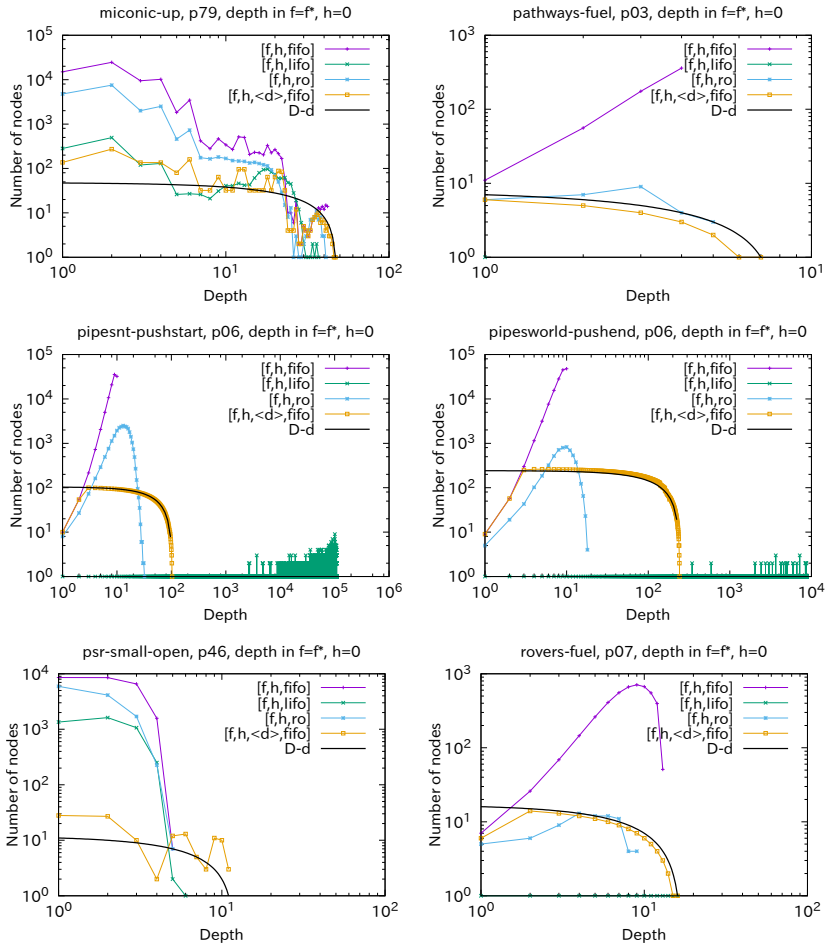

Figure 9.4.1: (Page 1/2) Number of nodes (y-axis) expanded per depth (x-axis) in the final plateau with different tie-breaking strategies. Both axes are in logarithmic scale.

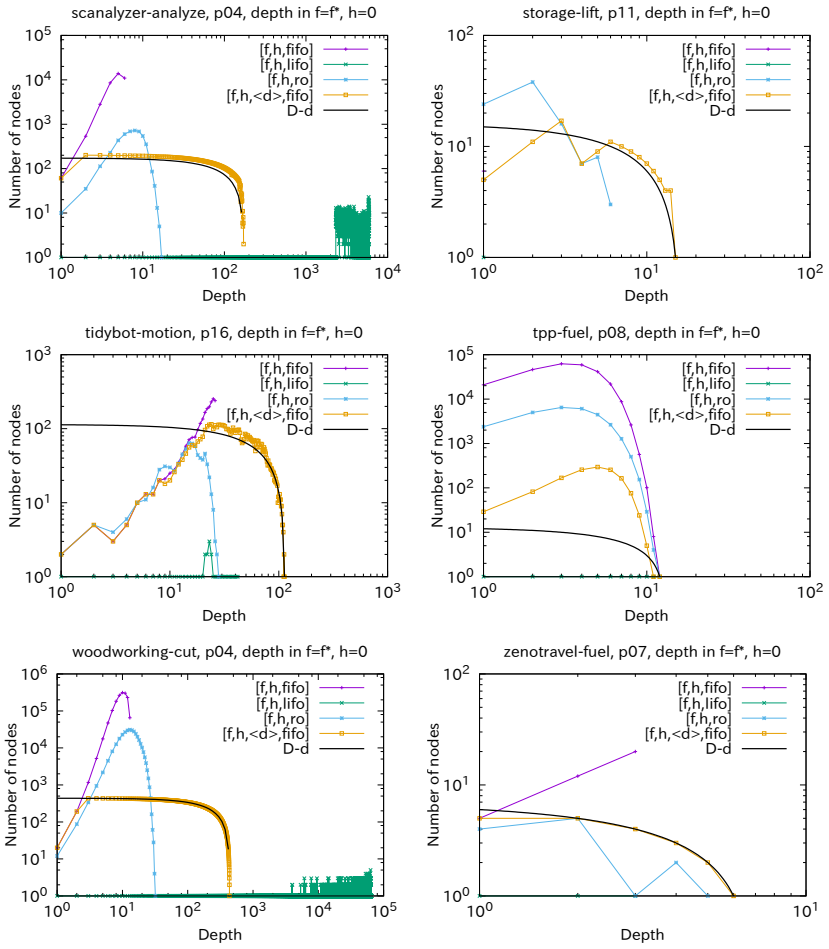

Figure 9.4.2: (Page 2/2) Number of nodes (y-axis) expanded per depth (x-axis) in the final plateau with different tie-breaking strategies. Both axes are in logarithmic scale.

9.5 Additional Figures for Figure 4.2.4: More Histograms for the Size of Non-final Plateaus

These are the additional histograms for the size of non-final plateaus on more variety of domains and instances.

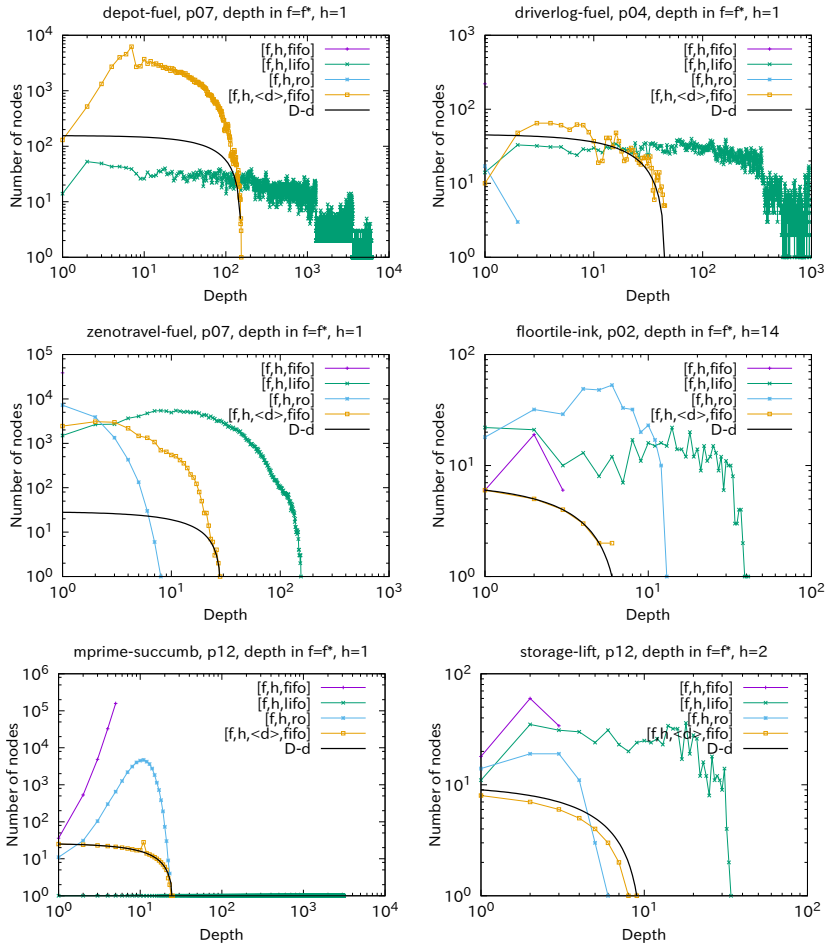

Figure 9.5.1: Depth distribution in the non-final plateaus (*plateau* (f^*, h), $h \neq 0$): Other domains.

9.6 Detailed Data for Table 5.5.2

	$[f, \hat{h}, \mathit{fifo}]$	$[f, \hat{h}, \mathit{lifo}]$	$[f, \hat{h}, \mathit{ro}]$	$[f, h, \hat{h}, \mathit{fifo}]$	$[f, h, \hat{h}, \mathit{lifo}]$	$[f, h, \hat{h}, \mathit{ro}]$	$[f, \hat{h}^{\mathrm{FF}}, \mathit{fifo}]$	$[f, \hat{h}^{\mathrm{FF}}, \mathit{lifo}]$	$[f, \hat{h}^{\mathrm{FF}}, \mathit{ro}]$	$[f, \hat{h}^{\mathrm{FF}}, \langle d \rangle, \mathit{fifo}]$	$[f, \hat{h}^{\mathrm{FF}}, \langle d \rangle, \mathit{lifo}]$	$[f, \hat{h}^{\mathrm{FF}}, \langle d \rangle, \mathit{ro}]$
Zerocost (620)	295	303	301.0	305	309	305.9 ± 2.1	337	340	341 ± 2.2	340	342	**344.3** ± 1.8
airport-fuel(20)	13	12	12.7	14	12	12.8 ± 0.8	13	11	11.7 ± 0.5	13	11	11.7 ± 0.5
blocks-stack(20)	15	15	15.0	15	15	15 ± 0	17	17	17 ± 0	17	17	17 ± 0
depot-fuel(22)	6	6	6.0	6	6	6 ± 0	6	6	6 ± 0	6	6	6 ± 0
driverlog-fuel(20)	8	8	8.0	8	8	8 ± 0	8	8	8 ± 0	8	8	8 ± 0
elevators-up(20)	**20**	**20**	19.9	**20**	**20**	**20** ± 0	**20**	**20**	**20** ± 0	**20**	**20**	**20** ± 0
floortile-ink(20)	8	8	8.0	8	8	8 ± 0	9	8	8.7 ± 0.5	9	8	8.7 ± 0.5
freecell-move(20)	12	14	13.3	12	14	13.2 ± 0.4	17	18	17.9 ± 0.8	17	18	18.3 ± 0.9
grid-fuel(5)	1	1	1.0	1	1	1 ± 0	1	1	1 ± 0	1	1	1 ± 0
gripper-move(20)	6	6	6.0	6	6	6 ± 0	6	6	6 ± 0	6	6	6 ± 0
hiking-fuel(20)	8	8	8.0	8	8	8 ± 0	9	9	9 ± 0	9	9	9 ± 0
logistics00-fuel(28)	15	15	15.0	15	15	15 ± 0	15	15	15 ± 0	15	15	15 ± 0
miconic-up(30)	14	17	15.1	14	17	15.1 ± 0.9	15	**21**	17.9 ± 1.2	15	**21**	18 ± 1.2
mprime-succumb(35)	19	16	19.1	20	16	20.1 ± 0.6	**30**	23	28.3 ± 0.9	**30**	27	29.3 ± 0.7
mystery-feast(20)	7	6	6.9	6	5	5.9 ± 0.3	**8**	**8**	**8** ± 0	**8**	**8**	**8** ± 0
nomystery-fuel(20)	10	10	10.0	10	10	10 ± 0	10	10	10 ± 0	10	10	10 ± 0
parking-movecc(20)	13	14	14.3	13	15	14.4 ± 1.5	**20**	**20**	**20** ± 0	**20**	**20**	**20** ± 0
pathways-fuel(30)	5	5	4.1	5	5	4 ± 0	5	5	5 ± 0	5	5	5 ± 0
pipesnt-pushstart(20)	7	8	7.7	8	8	7.8 ± 0.4	9	9	9 ± 0	9	9	9 ± 0
pipesworld-pushend(20)	5	6	5.1	5	5	5 ± 0	7	**8**	7.1 ± 0.3	7	7	7.7 ± 0.5
psr-small-open(20)	19	19	19.0	19	19	19 ± 0	19	19	19 ± 0	19	19	19 ± 0
rovers-fuel(40)	7	7	7.0	7	7	7 ± 0	8	9	8 ± 0	8	8	8 ± 0
scanalyzer-analyze(20)	8	11	10.1	16	**18**	15.3 ± 0.9	15	15	15 ± 0	15	15	15 ± 0
sokoban-pushgoal(20)	16	16	16.0	16	16	16 ± 0	17	17	17 ± 0	17	17	17 ± 0
storage-lift(20)	4	4	4.0	4	4	4 ± 0	4	4	4.3 ± 0.5	4	4	4.8 ± 0.4
tidybot-motion(20)	14	14	14.0	14	14	14 ± 0	15	16	16 ± 0	16	16	15.9 ± 0.3
tpp-fuel(30)	8	10	8.7	8	10	8.2 ± 0.4	8	10	9.1 ± 0.3	10	10	10 ± 0
woodworking-cut(20)	**20**	**20**	**20.0**	**20**	**20**	**20** ± 0	19	**20**	**20** ± 0	19	**20**	**20** ± 0
zenotravel-fuel(20)	7	7	7.0	7	7	7 ± 0	7	7	7 ± 0	7	7	7 ± 0

Table 9.6.1: Coverage results with LMcut for computing f and inadmissible distance-to-go heuristics for tie-breaking, on 620 Zerocost instances. We highlight the best results when the difference between the maximum and the minimum coverage exceeds 2, over all configurations *including Table 9.2.1*.

	$[f, \hat{h}, fifo]$	$[f, \hat{h}, lifo]$	$[f, \hat{h}, ro]$	$[f, h, \hat{h}, fifo]$	$[f, h, \hat{h}, lifo]$	$[f, h, \hat{h}, ro]$	$[f, \hat{h}^{\text{FF}}, fifo]$	$[f, \hat{h}^{\text{FF}}, lifo]$	$[f, \hat{h}^{\text{FF}}, ro]$	$[f, \hat{h}^{\text{FF}}, \langle d \rangle, fifo]$	$[f, \hat{h}^{\text{FF}}, \langle d \rangle, lifo]$	$[f, \hat{h}^{\text{FF}}, \langle d \rangle, ro]$
Zerocost (620)	308	305	307.3 ± 1.5	307	306	307.8 ± 1.4	336	331	**337.9** ± 2.1	337	333	337.6 ± 1.3
airport-fuel(20)	1	1	1 ± 0	1	1	1 ± 0	5	5	5 ± 0	5	5	5 ± 0
blocks-stack(20)	20	20	20 ± 0	20	20	20 ± 0	20	19	19.9 ± 0.3	20	20	19.9 ± 0.3
depot-fuel(22)	6	6	6 ± 0	6	6	6 ± 0	4	4	4 ± 0	4	4	4 ± 0
driverlog-fuel(20)	9	9	9 ± 0	9	9	9 ± 0	9	9	9 ± 0	9	9	9 ± 0
elevators-up(20)	19	19	19 ± 0	19	19	19 ± 0	**20**	**20**	**20** ± 0	**20**	**20**	**20** ± 0
floortile-ink(20)	8	8	8 ± 0	8	8	8 ± 0	9	8	8.8 ± 0.4	9	8	8.8 ± 0.4
freecell-move(20)	13	14	12.7 ± 0.7	13	13	12.7 ± 0.7	17	17	17.4 ± 0.5	17	17	17.3 ± 0.7
grid-fuel(5)	2	2	2 ± 0	2	2	2 ± 0	2	2	2 ± 0	2	2	2 ± 0
gripper-move(20)	20	20	20 ± 0	20	20	20 ± 0	20	20	20 ± 0	20	20	20 ± 0
hiking-fuel(20)	13	13	12.1 ± 0.3	13	13	12.1 ± 0.3	11	11	11 ± 0	11	11	11 ± 0
logistics00-fuel(28)	16	16	16 ± 0	16	16	16 ± 0	16	16	16 ± 0	16	16	16 ± 0
miconic-up(30)	22	22	22 ± 0	22	22	22.1 ± 0.3	30	30	30 ± 0	30	30	30 ± 0
mprime-succumb(35)	21	17	20.4 ± 0.7	21	17	20.4 ± 0.7	**28**	23	27.4 ± 0.7	**28**	25	27.7 ± 0.7
mystery-feast(20)	5	5	5 ± 0	5	5	5 ± 0	3	3	3 ± 0	3	3	3 ± 0
nomystery-fuel(20)	16	16	16 ± 0	16	16	16 ± 0	15	15	15 ± 0	15	15	15 ± 0
parking-movecc(20)	2	2	2 ± 0	2	2	2 ± 0	10	10	**10.3** ± 1.0	10	10	**10.3** ± 1.0
pathways-fuel(30)	4	4	4 ± 0	4	4	4 ± 0	4	4	4 ± 0	4	4	4 ± 0
pipesnt-pushstart(20)	1	2	1.9 ± 0.8	1	2	1.8 ± 0.7	5	5	5 ± 0	5	5	5 ± 0
pipesworld-pushend(20)	8	7	7.8 ± 0.4	8	8	8 ± 0	5	5	5.4 ± 0.7	5	5	5.6 ± 0.5
psr-small-open(20)	19	19	19 ± 0	19	19	19 ± 0	19	19	19 ± 0	19	19	19 ± 0
rovers-fuel(40)	8	8	8 ± 0	8	8	8 ± 0	8	8	8 ± 0	8	8	8 ± 0
scanalyzer-analyze(20)	15	14	15 ± 0	14	15	15 ± 0	15	16	**15.4** ± 0.7	15	15	15.2 ± 0.7
sokoban-pushgoal(20)	17	17	17 ± 0	17	17	17 ± 0	18	18	18.2 ± 0.4	18	18	18 ± 0
storage-lift(20)	4	4	4 ± 0	4	4	4 ± 0	4	4	4 ± 0	4	4	4 ± 0
tidybot-motion(20)	0	0	0 ± 0	0	0	0 ± 0	0	0	0 ± 0	0	0	0 ± 0
tpp-fuel(30)	9	10	9.4 ± 0.5	9	10	9.8 ± 0.4	10	11	10.9 ± 0.3	11	11	10.9 ± 0.3
woodworking-cut(20)	**20**	**20**	**20** ± 0	**20**	**20**	**20** ± 0	**20**	**20**	**20** ± 0	**20**	**20**	**20** ± 0
zenotravel-fuel(20)	10	10	10 ± 0	10	10	9.9 ± 0.3	9	9	9 ± 0	9	9	8.9 ± 0.3

Table 9.6.2: Coverage results with M&S for computing f and inadmissible distance-to-go heuristics for tie-breaking, on 620 Zerocost instances. We highlight the best results when the difference between the maximum and the minimum coverage exceeds 2, over all configurations *including Table 9.2.2*.

	$[f, \hat{h}, fifo]$	$[f, \hat{h}, lifo]$	$[f, \hat{h}, ro]$	$[f, h, \hat{h}, fifo]$	$[f, h, \hat{h}, lifo]$	$[f, h, \hat{h}, ro]$	$[f, \hat{h}^{\text{FF}}, fifo]$	$[f, \hat{h}^{\text{FF}}, lifo]$	$[f, \hat{h}^{\text{FF}}, ro]$	$[f, \hat{h}^{\text{FF}}, \langle d \rangle, fifo]$	$[f, \hat{h}^{\text{FF}}, \langle d \rangle, lifo]$	$[f, \hat{h}^{\text{FF}}, \langle d \rangle, ro]$
IPC benchmark (1104)	534	534	534 ± 2.1	536	535	534.7 ± 1.5	564	562	563.7 ± 1.4	563	560	561.9 ± 1.4
airport(50)	24	25	23.9 ± 0.6	24	24	23.8 ± 0.4	25	24	24.8 ± 0.4	25	24	24.6 ± 0.5
barman-opt11(20)	0	0	0 ± 0	0	0	0 ± 0	0	0	0 ± 0	0	0	0 ± 0
blocks(35)	27	27	27 ± 0	27	27	27 ± 0	27	27	27 ± 0	27	27	27 ± 0
cybersec(19)	5	3	5.9 ± 1.2	6	4	5.4 ± 0.7	6	6	5.9 ± 0.8	6	5	5.6 ± 0.7
depot(22)	5	5	5 ± 0	5	5	5 ± 0	6	6	6 ± 0	6	6	6 ± 0
driverlog(20)	12	12	12 ± 0	12	12	12 ± 0	13	13	13 ± 0	13	13	13 ± 0
elevators-opt11(20)	12	12	12 ± 0	12	12	12 ± 0	15	15	14.9 ± 0.3	14	15	14 ± 0
floortile-opt11(20)	6	6	6 ± 0	6	6	6 ± 0	6	6	6 ± 0	6	6	6 ± 0
freecell(80)	8	8	8 ± 0	8	8	8 ± 0	9	9	9 ± 0	9	9	9 ± 0
grid(5)	1	1	1 ± 0	1	1	1 ± 0	1	1	1 ± 0	1	1	1 ± 0
gripper(20)	6	6	6 ± 0	6	6	6 ± 0	6	6	6 ± 0	6	6	6 ± 0
hanoi(30)	11	11	11 ± 0	11	11	11 ± 0	12	12	12 ± 0	12	12	11.9 ± 0.3
logistics00(28)	17	17	17 ± 0	17	17	17 ± 0	20	20	20 ± 0	20	20	20 ± 0
miconic(150)	140	140	140 ± 0	140	140	140 ± 0	140	140	140 ± 0	140	140	140 ± 0
mprime(35)	20	21	19.9 ± 0.8	20	21	20 ± 0.7	22	22	22 ± 0	22	22	22 ± 0
mystery(30)	15	15	15 ± 0	15	15	15 ± 0	16	16	16 ± 0	16	16	16 ± 0
nomystery-opt11(20)	13	13	13 ± 0	13	13	13 ± 0	14	14	14 ± 0	14	14	14 ± 0
openstacks-opt11(20)	10	10	10 ± 0	10	10	9.9 ± 0.3	17	17	17 ± 0	17	17	17 ± 0
parcprinter-opt11(20)	13	13	13 ± 0	13	13	13 ± 0	13	13	13 ± 0	13	13	13 ± 0
parking-opt11(20)	1	1	1 ± 0	1	1	1 ± 0	1	1	1 ± 0	1	1	1 ± 0
pathways(30)	5	5	5 ± 0	5	5	5 ± 0	5	5	5 ± 0	5	5	5 ± 0
pegsol-opt11(20)	16	16	16 ± 0	16	16	16 ± 0	17	17	17 ± 0	17	17	17 ± 0
pipesworld-notankage(50)	12	12	12 ± 0	12	12	12 ± 0	13	13	13 ± 0	13	13	13 ± 0
pipesworld-tankage(50)	7	7	7 ± 0	7	7	7 ± 0	8	8	8 ± 0	8	8	8 ± 0
psr-small(50)	48	48	47.9 ± 0.3	48	48	48 ± 0	48	48	48 ± 0	48	48	48 ± 0
rovers(40)	7	7	7 ± 0	7	7	7 ± 0	7	7	7 ± 0	7	7	7 ± 0
scanalyzer-opt11(20)	8	10	8.8 ± 0.4	10	10	10 ± 0	10	10	10 ± 0	10	10	10 ± 0
sokoban-opt11(20)	17	17	17 ± 0	17	17	17 ± 0	19	19	19 ± 0	19	19	19 ± 0
storage(30)	14	14	14 ± 0	14	14	14 ± 0	14	14	14 ± 0	14	14	14 ± 0
tidybot-opt11(20)	10	11	10.3 ± 0.5	11	11	10.6 ± 0.5	11	11	11 ± 0	11	11	11 ± 0
tpp(30)	6	6	6 ± 0	6	6	6 ± 0	6	6	6 ± 0	6	6	6 ± 0
transport-opt11(20)	6	6	6 ± 0	6	6	6 ± 0	6	6	6 ± 0	6	6	6 ± 0
visitall-opt11(20)	10	10	10 ± 0	10	10	10 ± 0	10	10	10 ± 0	10	10	10 ± 0
woodworking-opt11(20)	11	8	9.3 ± 1.0	9	9	9 ± 0	10	9	10.1 ± 1.1	10	8	9.9 ± 1.1
zenotravel(20)	11	11	11 ± 0	11	11	11 ± 0	11	11	11 ± 0	11	11	11 ± 0

Table 9.6.3: Coverage results with LMcut for computing f and inadmissible distance-to-go heuristics for tie-breaking, on 1104 standard IPC benchmark instances.

	$[f, \hat{h}, fifo]$	$[f, \hat{h}, lifo]$	$[f, \hat{h}, ro]$	$[f, h, \hat{h}, fifo]$	$[f, h, \hat{h}, lifo]$	$[f, h, \hat{h}, ro]$	$[f, \hat{h}^{\mathrm{FF}}, fifo]$	$[f, \hat{h}^{\mathrm{FF}}, lifo]$	$[f, \hat{h}^{\mathrm{FF}}, ro]$	$[f, \hat{h}^{\mathrm{FF}}, \langle d \rangle, fifo]$	$[f, \hat{h}^{\mathrm{FF}}, \langle d \rangle, lifo]$	$[f, \hat{h}^{\mathrm{FF}}, \langle d \rangle, ro]$
IPC benchmark (1104)	477	475	470.4 ± 0.9	476	475	470.9 ± 0.9	458	457	457 ± 1.3	457	457	456.8 ± 1.2
airport(50)	7	7	7 ± 0	7	7	7 ± 0	9	9	9 ± 0	9	9	9 ± 0
barman-opt11(20)	4	4	4 ± 0	4	4	4 ± 0	4	4	4 ± 0	4	4	4 ± 0
blocks(35)	22	21	21 ± 0	21	21	21 ± 0	21	20	20.1 ± 0.3	20	20	20 ± 0
cybersec(19)	0	0	0 ± 0	0	0	0 ± 0	0	0	0 ± 0	0	0	0 ± 0
depot(22)	5	5	5 ± 0	5	5	5 ± 0	4	4	4 ± 0	4	4	4 ± 0
driverlog(20)	12	12	12 ± 0	12	12	12 ± 0	11	11	11 ± 0	11	11	11 ± 0
elevators-opt11(20)	13	13	12 ± 0	13	13	12 ± 0	10	10	10 ± 0	10	10	10 ± 0
floortile-opt11(20)	6	6	6 ± 0	6	6	6 ± 0	7	7	7 ± 0	7	7	7 ± 0
freecell(80)	15	15	15 ± 0	15	15	15 ± 0	14	14	14 ± 0	14	14	14 ± 0
grid(5)	2	2	2 ± 0	2	2	2 ± 0	2	2	2 ± 0	2	2	2 ± 0
gripper(20)	20	20	20 ± 0	20	20	20 ± 0	20	20	20 ± 0	20	20	20 ± 0
hanoi(30)	14	14	14 ± 0	14	14	14 ± 0	13	13	13 ± 0	13	13	13 ± 0
logistics00(28)	20	20	20 ± 0	20	20	20 ± 0	20	20	20 ± 0	20	20	20 ± 0
miconic(150)	72	72	72 ± 0.5	72	72	72 ± 0.5	69	69	69.2 ± 0.4	69	69	69.2 ± 0.4
mprime(35)	19	19	19.3 ± 0.5	20	19	19.3 ± 0.5	21	21	21.1 ± 0.8	21	21	21.2 ± 0.7
mystery(30)	15	15	15 ± 0	15	15	15 ± 0	15	15	15 ± 0	15	15	15 ± 0
nomystery-opt11(20)	18	18	18 ± 0	18	18	18 ± 0	16	16	16 ± 0	16	16	16 ± 0
openstacks-opt11(20)	18	19	18 ± 0	18	19	18 ± 0	18	18	18 ± 0	18	18	17.7 ± 0.5
parcprinter-opt11(20)	10	10	10 ± 0	10	10	10 ± 0	11	11	11 ± 0	11	11	11 ± 0
parking-opt11(20)	1	1	0.6 ± 0.5	1	1	0.8 ± 0.4	0	0	0 ± 0	0	0	0 ± 0
pathways(30)	4	4	4 ± 0	4	4	4 ± 0	4	4	4 ± 0	4	4	4 ± 0
pegsol-opt11(20)	19	19	19 ± 0	19	19	19 ± 0	17	17	17 ± 0	17	17	17 ± 0
pipesworld-notankage(50)	6	5	5.7 ± 0.7	6	5	5.9 ± 0.8	9	9	8.7 ± 0.5	9	9	8.8 ± 0.4
pipesworld-tankage(50)	12	12	12 ± 0	12	12	12 ± 0	9	9	9 ± 0	9	9	9 ± 0
psr-small(50)	50	50	50 ± 0	50	50	50 ± 0	50	50	50 ± 0	50	50	50 ± 0
rovers(40)	8	8	6 ± 0	7	8	6.1 ± 0.3	6	6	6 ± 0	6	6	6 ± 0
scanalyzer-opt11(20)	10	10	9.9 ± 0.3	10	10	9.8 ± 0.4	7	7	6.8 ± 0.4	7	7	6.8 ± 0.4
sokoban-opt11(20)	18	18	18 ± 0	18	18	18 ± 0	19	19	19 ± 0	19	19	19 ± 0
storage(30)	15	15	15 ± 0	15	15	15 ± 0	14	14	14 ± 0	14	14	14 ± 0
tidybot-opt11(20)	0	0	0 ± 0	0	0	0 ± 0	0	0	0 ± 0	0	0	0 ± 0
tpp(30)	6	6	6 ± 0	6	6	6 ± 0	6	6	6 ± 0	6	6	6 ± 0
transport-opt11(20)	7	7	6 ± 0	7	7	6 ± 0	6	6	6 ± 0	6	6	6 ± 0
visitall-opt11(20)	9	9	9 ± 0	9	9	9 ± 0	9	9	9 ± 0	9	9	9 ± 0
woodworking-opt11(20)	8	8	8.1 ± 0.3	8	8	8.1 ± 0.3	7	7	7.1 ± 0.3	7	7	7.1 ± 0.3
zenotravel(20)	12	11	10.9 ± 0.3	12	11	10.9 ± 0.3	10	10	10 ± 0	10	10	10 ± 0

Table 9.6.4: Coverage results with M&S for computing f and inadmissible distance-to-go heuristics for tie-breaking, on 1104 standard IPC benchmark instances.

Bibliography

Aghighi, M., & Bäckström, C. (2015). Cost-Optimal and Net-Benefit Planning– A Parameterised Complexity View. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Aghighi, M., & Bäckström, C. (2016). A Multi-Parameter Complexity Analysis of Cost-Optimal and Net-Benefit Planning. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Amir, E., & Engelhardt, B. (2003). Factored Planning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Asai, M., & Fukunaga, A. (2015). Solving Large-Scale Planning Problems by Decomposition and Macro Generation. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*, Jerusalem, Israel.

Asai, M., & Fukunaga, A. (2016). Tiebreaking Strategies for A^* Search: How to Explore the Final Frontier. In *Proceedings of AAAI Conference on Artificial Intelligence*, Arizona, USA.

Asai, M., & Fukunaga, A. (2017a). Exploration Among and Within Plateaus in Greedy Best-First Search. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*, Pittsburgh, USA.

Asai, M., & Fukunaga, A. (2017b). Tie-Breaking Strategies for Cost-Optimal Best First Search. *J. Artif. Intell. Res.(JAIR)*, *58*, 67–121.

Barabási, A.-L. (1996). Invasion Percolation and Global Optimization. *Physical Review Letters*, *76*(20), 3750.

Benton, J., Talamadupula, K., Eyerich, P., Mattmüller, R., & Kambhampati, S. (2010). G-Value Plateaus: A Challenge for Planning. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Boddy, M. S., Gohde, J., Haigh, T., & Harp, S. A. (2005). Course of Action Generation for Cyber Security Using Classical Planning. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Bonet, B. (2013). An Admissible Heuristic for SAS+ Planning Obtained from the State Equation.. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Botea, A., Enzenberger, M., Müller, M., & Schaeffer, J. (2005). Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *J. Artif. Intell. Res.(JAIR)*, *24*, 581–621.

Brafman, R. I., & Domshlak, C. (2006). Factored Planning: How, When, and When Not. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Burns, E. (2012). Heuristic search code library. https://github.com/eaburns/search.

Burns, E. A., Hatem, M., Leighton, M. J., & Ruml, W. (2012). Implementing Fast Heuristic Search Code. In *Proceedings of Annual Symposium on Combinatorial Search*.

Chrpa, L., Vallati, M., & McCluskey, T. L. (2015). On the Online Generation of Effective Macro-Operators. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Chu, Y.-J., & Liu, T.-H. (1965). On Shortest Arborescence of a Directed Graph. *Scientia Sinica*, *14*(10), 1396.

Coles, A., & Smith, A. (2007). Marvin: A Heuristic Search Planner with Online Macro-Action Learning. *J. Artif. Intell. Res.(JAIR)*, *28*, 119–156.

Cushing, W., Benton, J., & Kambhampati, S. (2010). Cost Based Search Considered Harmful. In *Proceedings of Annual Symposium on Combinatorial Search*.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, *1*(1), 269–271.

Domshlak, C., Katz, M., & Shleyfman, A. (2013). Symmetry Breaking: Satisficing Planning and Landmark Heuristics. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Edmonds, J. (1967). Optimum Branchings. *Journal of Research of the National Bureau of Standards B*, *71*(4), 233–240.

Eyerich, P., Mattmüller, R., & Röger, G. (2009). Using the Context-Enhanced Additive Heuristic for Temporal and Numeric Planning. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Felner, A., Kraus, S., & Korf, R. E. (2003). KBFS: K-Best-First Search. *Annals of Mathematics and Artificial Intelligence*, *39*(1-2), 19–39.

Felner, A., Zahavi, U., Holte, R. C., Schaeffer, J., Sturtevant, N. R., & Zhang, Z. (2011). Inconsistent Heuristics in Theory and Practice. *J. Artif. Intell. Res.(JAIR)*, *175*(9-10), 1570–1603.

Fink, A., & Voss, S. (1999). Applications of Modern Heuristic Search Methods to Pattern Sequencing Problems. *Computers & Operations Research*, *26*(1), 17–34.

Fox, M., & Long, D. (1998). The Automatic Inference of State Invariants in TIM. *J. Artif. Intell. Res.(JAIR)*, 367–421.

Gabow, H. N., Galil, Z., Spencer, T., & Tarjan, R. E. (1986). Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graphs. *Combinatorica*, *6*(2), 109–122.

Gerevini, A. E., Saetti, A., & Vallati, M. (2009). An Automatically Configurable Portfolio-based Planner with Macro-actions: PbP. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Hall, D. L. W., Cohen, A., Burkett, D., & Klein, D. (2013). Faster Optimal Planning with Partial-Order Pruning. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Hansen, E. A., & Zhou, R. (2007). Anytime Heuristic Search. *J. Artif. Intell. Res.(JAIR)*, *28*, 267–297.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on*, *4*(2), 100–107.

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

Helmert, M. (2006). The Fast Downward Planning System. *J. Artif. Intell. Res.(JAIR)*, *26*, 191–246.

Helmert, M., & Domshlak, C. (2009). Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Helmert, M., Haslum, P., & Hoffmann, J. (2007). Flexible Abstraction Heuristics for Optimal Sequential Planning. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*, pp. 176–183.

Helmert, M., Haslum, P., Hoffmann, J., & Nissim, R. (2014). Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *Journal of ACM*, *61*(3), 16:1–16:63.

Heusner, M., Keller, T., & Helmert, M. (2017). Understanding the Search Behaviour of Greedy Best-First Search. In *Proceedings of Annual Symposium on Combinatorial Search*.

Hoffmann, J. (2005). Where 'Ignoring Delete Lists' Works: Local Search Topology in Planning Benchmarks. *J. Artif. Intell. Res.(JAIR)*, *24*, 685–758.

Hoffmann, J. (2011). Analyzing Search Topology Without Running Any Search: On the Connection Between Causal Graphs and h^+. *J. Artif. Intell. Res.(JAIR)*, *41*(2), 155–229.

Hoffmann, J., & Nebel, B. (2001). The FF Planning System: Fast Plan Generation through Heuristic Search. *J. Artif. Intell. Res.(JAIR)*, *14*, 253–302.

Holte, R. C. (2010). Common Misconceptions Concerning Heuristic Search. In *Proceedings of Annual Symposium on Combinatorial Search*.

Imai, T., & Kishimoto, A. (2011). A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Kautz, H. A., & Selman, B. (1992). Planning as Satisfiability.. In *Proceedings of European Conference on Artificial Intelligence*, Vol. 92, pp. 359–363.

Korf, R. E. (1985). Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, *27*(1), 97–109.

Korf, R. E. (1999). Divide-and-Conquer Bidirectional Search: First Results. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Korf, R. E., & Zhang, W. (2000). Divide-and-Conquer Frontier Search Applied to Optimal Sequence Alignment. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Lelis, L. H., Zilles, S., & Holte, R. C. (2013). Stratified Tree Search: A Novel Suboptimal Heuristic Search Algorithm. In *AAMAS*, pp. 555–562.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., & Thrun, S. (2008). Anytime search in dynamic graphs. *Artificial Intelligence*, *172*(14), 1613–1643.

McDermott, D. V. (2000). The 1998 AI Planning Systems Competition. *AI Magazine*, *21*(2), 35–55.

Monnerot-Dumaine, A. (2006). `https://commons.wikimedia.org/wiki/File:Amas_de_percolation.png` (CC BY-SA 3.0)..

Nakhost, H., & Müller, M. (2009). Monte-Carlo Exploration for Deterministic Planning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Nilsson, N. J. (1971). *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Pub. Co., Inc., Reading, MA.

Pearl, J., & Kim, J. H. (1982). Studies in Semi-Admissible Heuristics. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, pp. 392–399.

Pochter, N., Zohar, A., & Rosenschein, J. S. (2011). Exploiting Problem Symmetries in State-Based Planners. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Pohl, I. (1973). The Avoidance of (Relative) Catastrophe, Heuristic Competence, Genuine Dynamic Weighting and Computational Issues in Heuristic Problem Solving. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Prim, R. C. (1957). Shortest Connection Networks and Some Generalizations. *Bell System Technical Journal*, *36*(6), 1389–1401.

Richter, S., Helmert, M., & Westphal, M. (2008). Landmarks Revisited. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Richter, S., & Westphal, M. (2010). The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.(JAIR)*, *39*(1), 127–177.

Richter, S., Westphal, M., & Helmert, M. (2011). LAMA 2008 and 2011. In *Proceedings of International Planning Competition*, pp. 117–124.

Röger, G., & Helmert, M. (2010). The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Schulte, T., & Keller, T. (2014). Balancing Exploration and Exploitation in Classical Planning. In *Proceedings of Annual Symposium on Combinatorial Search*.

Sheppard, A. P., Knackstedt, M. A., Pinczewski, W. V., & Sahimi, M. (1999). Invasion percolation: New algorithms and universality classes. *Journal of Physics A: Mathematical and General*, *32*(49), L521.

Sievers, S., Ortlieb, M., & Helmert, M. (2012). Efficient Implementation of Pattern Database Heuristics for Classical Planning.. In *Proceedings of Annual Symposium on Combinatorial Search*.

Stern, R., Kulberis, T., Felner, A., & Holte, R. C. (2010). Using Lookaheads with Optimal Best-First Search. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Tarjan, R. E. (1977). Finding Optimum Branchings. *Networks*, *7*(1), 25–35.

Thayer, J. T., & Ruml, W. (2009). Using Distance Estimates in Heuristic Search. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Thayer, J. T., & Ruml, W. (2011). Bounded Suboptimal Search: A Direct Approach using Inadmissible Estimates. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Valenzano, R. A., Schaeffer, J., Sturtevant, N., & Xie, F. (2014). A Comparison of Knowledge-Based GBFS Enhancements and Knowledge-Free Exploration. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Valenzano, R. A., & Xie, F. (2016). On the Completeness of Best-First Search Variants that Use Random Exploration. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Vallati, M., Hutter, F., Chrpa, L., & McCluskey, T. L. (2015). On the Effective Configuration of Planning Domain Models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

van den Briel, M., & Kambhampati, S. (2005). Optiplan: Unifying IP-based and Graph-based Planning. *J. Artif. Intell. Res.(JAIR)*, *24*, 919–931.

Wehrle, M., Helmert, M., Alkhazraji, Y., & Mattmüller, R. (2013). The Relative Pruning Power of Strong Stubborn Sets and Expansion Core. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Wilkinson, D., & Willemsen, J. F. (1983). Invasion Percolation: A New Form of Percolation Theory. *Journal of Physics A: Mathematical and General*, *16*(14), 3365.

Wilt, C. M., & Ruml, W. (2011). Cost-Based Heuristic Search is Sensitive to the Ratio of Operator Costs. In *Proceedings of Annual Symposium on Combinatorial Search*.

Wilt, C. M., & Ruml, W. (2014). Speedy versus Greedy Search. In *Proceedings of Annual Symposium on Combinatorial Search*.

Wray, K. H., Zilberstein, S., & Mouaddib, A.-I. (2015). Multi-Objective MDPs with Conditional Lexicographic Reward Preferences. In *Proceedings of AAAI Conference on Artificial Intelligence*, pp. 3418–3424.

Xie, F., Müller, M., & Holte, R. (2014). Adding Local Exploration to Greedy Best-First Search in Satisficing Planning. In *Proceedings of AAAI Conference on Artificial Intelligence*, pp. 2388–2394.

Xie, F., Müller, M., & Holte, R. (2015). Understanding and Improving Local Exploration for GBFS. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*, pp. 244–248.

Xie, F., Müller, M., Holte, R. C., & Imai, T. (2014). Type-Based Exploration with Multiple Search Queues for Satisficing Planning. In *Proceedings of AAAI Conference on Artificial Intelligence*.