Tiebreaking Strategies for A* Search: How to Explore the Final Frontier

Masataro Asai and **Alex Fukunaga**
Graduate School of Arts and Sciences
The University of Tokyo

Abstract

Despite recent improvements in search techniques for cost-optimal classical planning, the exponential growth of the size of the search frontier in A* is unavoidable. We investigate tiebreaking strategies for A*, experimentally analyzing the performance of standard tiebreaking strategies that break ties according to the heuristic value of the nodes. We find that tiebreaking has a significant impact on search algorithm performance when there are zero-cost operators that induce large plateau regions in the search space. We develop a new framework for tiebreaking based on a depth metric which measures distance from the entrance to the plateau, and propose a new, randomized strategy which significantly outperforms standard strategies on domains with zero-cost actions.

1 Introduction

This paper investigates tiebreaking strategies for A^*, the 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, Nilsson, and Raphael 1968). In each iteration, A^* selects and *expands* a node n from the OPEN priority queue. n is the node which has the lowest f-cost in OPEN, where for node n, $f(n)$ is the sum of $g(n)$, the cost of the current path from the initial state to n, and $h(n)$, a heuristic estimate of the cost from n to a goal state. A^* returns an optimal solution when h is admissible, i.e., when $h \leq h^*$, where h^* is the optimal distance to the goal.

If f^* is the cost of the optimal solution, the *effective search space of* A^* is the set of nodes with $f(n) \leq f^*$, and much of the work in the search and planning literature has focused on reducing the size of this effective search space by developing more accurate, admissible heuristic functions.

In many problems, the size of the last layer of search (which explores the set of nodes with $f(n) = f^*$) accounts for a significant fraction of the effective search space of A^*. Figure 1 plots the number of states with $f(n) = f^*$ (y-axis) vs. the # of states with $f(n) \leq f^*$ for 1104 problem instances from the International Planning Competition (IPC1998-2011). For many instances, a large fraction of the nodes in the effective search space have $f(n) = f^*$. For example, in the Openstacks domain, almost all states with $f(n) \leq f^*$ have cost f^* due to the large number of actions with cost 0. In such domains, the tiebreaking policy which decides which nodes to expand in the final frontier can have a significant impact on the performance of A^*.

In this paper, we investigate the *tiebreaking* strategy used by A^*, which is the policy for selecting which node to expand among nodes with the same f-cost. 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 tiebreaking requires more careful consideration, particularly for problems with large *plateaus* – regions of the search space with the same f and h values.

We first empirically evaluate standard tiebreaking strategies for A^*, and show that (1) a Last-In-First-Out (*lifo*) policy tends to be more efficient than a First-In-First-Out (*fifo*) policy, and (2) tiebreaking according to the heuristic value h, which frequently appears in the heuristic search literature, has little impact on the performance as long as a *lifo* policy is used. We show that there are significant performance differences among tiebreaking strategies when domains include zero-cost actions. While there are relatively few domains with zero-cost actions in the IPC benchmark set, we argue that zero-cost actions naturally occur in practical cost-minimization problems.

In order to solve such problems more efficiently, we propose tiebreaking methods based on a notion of *depth* within the plateau, corresponding to the number of steps a node is from the "entrance" to the plateau. We empirically show that: (1) a randomized, depth-based strategy significantly outperforms other tiebreaking strategies using the same heuristic function; (2) although depth is a component of a multi-level tiebreaking strategy, the depth is the principal factor in determining performance; and (3) depth-based tiebreaking is robust, in the sense that it does not rely on a particular action ordering in the domain definition. Note that all tiebreaking strategies in this paper maintain the optimality of the search algorithm because they only affect node expansion order among the nodes with the same f-cost.

We report our results using landmark-cut (LMcut) (Helmert and Domshlak 2009) and merge-and-shrink (Helmert et al. 2014) heuristics. Due to space, detailed results are only provided for LMcut. A more complete, de-

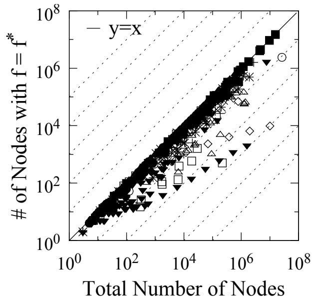

Figure 1: The # of nodes with $f = f^*$ (y-axis) compared to the total # of nodes in the search space (x-axis) with $f \leq f^*$ on 1104 IPC benchmark problems, using modified Fast Downward with LMcut which generates all nodes with cost f^*.

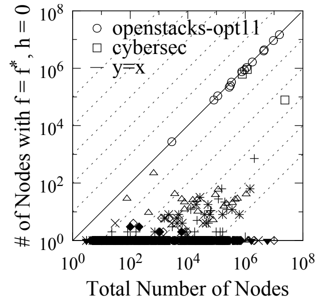

Figure 2: Similar to Figure 1; y-axis shows # nodes with $f = f^*, h = 0$, which forms the final plateau when h-based tiebreaking is enabled. Note that many Openstacks and Cybersec instances are near the $y = x$ line.

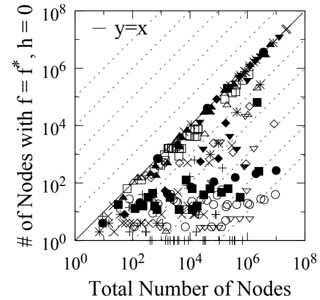

Figure 3: Similar to Figure 2, but for 620 instances from our *zerocost domains* (Sec. 5), where zero-cost actions induce very large plateaus.

tailed set of results, including color figures, is available at the author's website (http://guicho271828.github.io/publications/).

2 Preliminaries and Definitions

We first define some notation and terminology used throughout the rest of the paper. A *tiebreaking strategy* selects from among nodes with the same f-value. Tiebreaking strategies are denoted as $[\text{criterion}_1, \text{criterion}_2, ..., \text{criterion}_k]$, which means: If there are multiple nodes with the same f-value, first, break ties using criterion_1. If there are still multiple nodes remaining, then break ties using criterion_2 and so on, until a single node is selected. The *first-level tiebreaking policy* of a strategy is criterion_1, the *second-level tiebreaking policy* is criterion_2, and so on.

A *plateau* is a set of nodes in OPEN with both the same f and same h costs. A plateau whose nodes have f-cost f_p and h-cost h_p is denoted as *plateau* (f_p, h_p). An *entrance* to a *plateau* (f_p, h_p) is a node $n \in$ *plateau* (f_p, h_p), whose current parent is not a member of *plateau* (f_p, h_p). The *final plateau*, is the plateau containing the solution found by the search algorithm. In A^* using admissible heuristics, the final plateau is *plateau* $(f^*, 0)$.

3 Background: Tiebreaking Strategies in A^*

If multiple nodes with the same f-cost are possible, A^* must implement some tiebreaking policy (either explicitly or implicitly) which selects from among these nodes. The early literature on heuristic search seems to have been mostly agnostic regarding tiebreaking. 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, Nilsson, and Raphael 1968, p.102 Step 2), (Nilsson 1971, p.69). Pearl's textbook on heuristic search specifies that best-first search should "break ties arbitrarily" (1984, p.48, Step 3), and does not specifically mention tiebreaking for A^*. To the best of our knowledge, the first explicit mention of a tiebreaking policy that considers node generation order is by Korf in his analysis of IDA*: "If A^* employs the tiebreaking rule of 'most-recently generated', it must also expand the same nodes [as IDA*]", i.e., a *lifo* ordering.

In recent years, tiebreaking according to h-values has become "folklore" in the search community. Hansen and Zhou state that "It is well-known that A^* achieves best performance when it breaks ties in favor of nodes with least h-cost" (Hansen and Zhou 2007). Holte writes "A^* breaks ties in favor of larger g-values, as is most often done" (Holte 2010, note that since $f = g + h$, preferring large g is equivalent to preferring smaller h). In their detailed survey/tutorial on efficient A^* implementations, Burns et al. (2012) also break ties "preferring high g" (equivalent to low h). Thus, tiebreaking according to h-values appears to be ubiquitous in practice. To our knowledge, an in-depth, experimental analysis of tiebreaking strategies for A^* is lacking in the literature.

Although the standard practice of tiebreaking according to h might be sufficient in some domains, further levels of tiebreaking (explicit or implicit) are required if multiple nodes can have the same f and h values. While the survey of efficient A^* implementation techniques in (Burns et al. 2012) did not explicitly mention 2nd-level tiebreaking, their library code (https://github.com/eaburns/search) first breaks ties according to h, and then breaks remaining ties according to a *lifo* policy (most recently generated nodes first), i.e., a $[h, \textit{lifo}]$ strategy. Although not documented, their choice of a *lifo* 2nd-level tiebreaking policy appears to be a natural consequence of the fact it can be trivially, 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 $[h, \textit{fifo}]$ tiebreaking strategy. Although we could not find an explanation, this choice is most likely due to their use of alternating OPEN lists, in which case the *fifo* second-level policy serves to provide a limited form of fairness.

4 Evaluation of Standard Strategies

We evaluated tiebreaking strategies for domain-independent, classical planning. In our experiments, the planners are

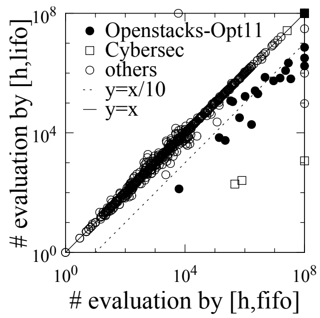

Figure 4: # of evaluations of standard *fifo* vs *lifo* second-level tiebreaking, with first-level h tiebreaking. *lifo* evaluates less than $1/10$ of the nodes evaluated by *fifo* in Cybersec and Openstacks.

based on Fast Downward (revision 6251), and all experiments are run with a 5-minute, 2GB 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. We used 1104 instances from 35 standard benchmark domains.

We first compared two commonly used tiebreaking strategies, $[h,$*fifo*$]$, $[h,$*lifo*$]$, which first break ties according to h, and then apply *fifo* or *lifo* second-level tiebreaking, respectively. Detailed results for the LMcut heuristic (Helmert and Domshlak 2009), as well as summary results for the M&S heuristic (Helmert et al. 2014), are shown in Table 1 (leftmost 2 columns). Differences in coverage are observed in several domains, and $[h,$*lifo*$]$ outperforms $[h,$*fifo*$]$ in total. Figure 4 gives us a more fine-grained analysis by comparing the number of node evaluation (computations of LMcut) of the $[h,$*lifo*$]$ and $[h,$*fifo*$]$ strategies. It shows that the difference in the # of nodes evaluated can sometimes be larger than a factor of 10 (Openstacks, Cybersec domains).

Is h-Based Tiebreaking Necessary? Table 1 also shows the results of [*fifo*] and [*lifo*], which rely only on *fifo* or *lifo* tiebreaking. [*lifo*], which simply breaks ties among nodes with the same f-cost by expanding most recently generated nodes first (Korf 1985), clearly dominates [*fifo*].

Interestingly, the performance of the [*lifo*] strategy is comparable to $[h,$*lifo*$]$ and $[h,$*fifo*$]$, the standard two-level strategies that first break ties according to h. This may be surprising, considering the ubiquity of h-based tiebreaking in the search and planning communities. However, *lifo* behaves somewhat similarly to h-based tiebreaking, in the following sense: *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 $h(n') = h(n)$, because $g(n)+h(n) = g(n')+h(n')$. Thus, as *lifo* expands nodes in a "depth-first" manner, the nodes that continue to be expanded by *lifo*'s depth-first exploration have non-increasing h-values, much like in h-based tiebreaking. Although in general, the expansion order of [*lifo*] is not the same as that of h-based tiebreaking strategies, this might explain why their performances are comparable. An in-depth investigation of the behavior of [*lifo*] vs. h-based tiebreaking is a direction for future work.

Plateaus and Tiebreaking In Figure 4, we observed that large performance differences between 2-level tiebreaking strategies $[h,$*lifo*$]$ and $[h,$*fifo*$]$ tend to occur in problems where there are many nodes with the same f and h values, creating large plateau regions where the heuristic does not provide any useful guidance – by definition, these plateau regions require a blind search (because all nodes have the same f, h) which relies solely on the tiebreaking criterion.

Figure 2 plots the size of the final plateau on 1104 IPC benchmark instances. The y-axis represents the # of nodes with $f = f^*, h = 0$, i.e., the final plateau, and the x-axis represents the total # of nodes with $f \leq f^*$. In some domains such as Openstacks and Cybersec, the planner spends most of the runtime searching the final plateau even with h tiebreaking, and thus the runtime on these domains varies significantly depending on the second-level tiebreaking strategy.

5 Domains with Zero-Cost Actions

Openstacks is a cost minimization domain introduced in IPC-2006, where the objective is to minimize the number of stacks used. There are many zero-cost actions (i.e., actions that don't increase the number of stacks), and they prevent the standard heuristics from producing informative guidance.

Although domains with zero-cost actions are not common in the current set of benchmarks, we argue that such domains are of an important class of models for cost-minimization problems, i.e., assigning zero costs make 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.

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 difficult for standard tiebreaking methods in the previous section, both contain many zero-cost actions, and both are based on industrial applications: Openstacks models production planning (Fink and Voss 1999) and Cybersec models Behavioral Adversary Modeling System (Boddy et al. 2005, minimizing decryption, data transfer, etc.).

Therefore, in this paper, we modified various domains into cost minimization domains with many zero-cost actions. Specifically, the domain is modified so that all action schemas are assigned cost 0 except for 1 action schema which consumes some key resource. The last word in the names of these domains indicate the action which is assigned non-zero cost, e.g., elevator-up is a modified elevator domain where the up action is assigned non-zero cost, and all other actions have 0 cost. Most of the transportation-type domains are modified to optimize energy usage (Logistics-

	Coverages (# problems solved)				Coverage (# problems solved), 10 runs (mean±sd)					Wilcoxon p vs $[h, rd, ro]$		
Domain	$[h, fifo]$	$[h, lifo]$	$[fifo]$	$[lifo]$	$[h, fd, ro]$	$[h, ld, ro]$	$[h, rd, ro]$	$[rd, ro]$	$[h, ro]$	$[h, fd, ro]$	$[h, ld, ro]$	$[h, ro]$
LMcut IPC (1104)	558	565	442	556	556.6±0.7	570.3±2.1	**572.8±0.7**	558.8±2.1	559.8±1.0	**0.0**	**.01**	**0.0**
airport(50)	**27**	26	18	26	26.2±0.4	26.2±0.4	26.2±0.4	21.0±0.0	26.0±0.0	1.0	1.0	.17
cybersec(19)	2	3	0	3	2.0±0.0	8.5±2.0	**10.9±0.8**	7.4±0.7	4.4±1.0	**0.0**	**.01**	**0.0**
logistics00(28)	**20**	**20**	16	18	**20.0±0.0**	**20.0±0.0**	**20.0±0.0**	**20.0±0.0**	**20.0±0.0**	1.0	1.0	1.0
miconic(150)	**140**	**140**	68	**140**	**140.0±0.0**	**140.0±0.0**	**140.0±0.0**	135.5±1.2	**140.0±0.0**	1.0	1.0	1.0
openstacks-opt11(20)	11	**18**	11	**18**	11.0±0.0	**18.0±0.0**	**18.0±0.0**	**18.0±0.0**	11.6±0.5	**0.0**	1.0	**0.0**
pipesworld-notankage(50)	**15**	14	13	13	14.4±0.5	14.6±0.5	14.7±0.5	14.3±0.5	14.9±0.3	0.2	.68	0.3
scanalyzer-opt11(20)	**10**	**10**	4	**10**	**10.0±0.0**	**10.0±0.0**	**10.0±0.0**	9.0±0.0	**10.0±0.0**	1.0	1.0	1.0
woodworking-opt11(20)	10	10	6	9	10.0±0.0	10.0±0.0	10.0±0.0	**11.6±0.5**	10.0±0.0	1.0	1.0	1.0
LMcut Zerocost(620)	256	279	212	281	257.4±2.0	286.6±7.1	**294.2±2.3**	279.9±3.9	264.9±1.8	**0.0**	**.01**	**0.0**
airport-fuel(20)	**15**	13	7	**15**	14.7±1.0	14.0±0.6	14.6±0.5	10.5±0.7	14.4±0.7	.59	**.05**	.58
driverlog-fuel(20)	**8**	**8**	7	**8**	**8.0±0.0**	7.7±0.5	**8.0±0.0**	**8.0±0.0**	**8.0±0.0**	1.0	.08	1.0
elevators-up(20)	7	**13**	7	**13**	7.0±0.0	9.4±0.7	10.7±1.1	8.3±0.6	7.3±0.5	**0.0**	**.02**	**0.0**
freecell-move(20)	4	19	4	19	4.0±0.0	**19.7±0.5**	17.2±0.6	16.7±1.0	5.0±0.4	**0.0**	**0.0**	**0.0**
miconic-up(30)	16	17	10	17	15.7±0.5	19.4±0.7	**20.4±1.2**	**20.4±0.9**	17.0±0.4	**0.0**	**.03**	**0.0**
mprime-succumb(35)	15	14	12	14	16.3±0.5	18.9±4.0	**20.5±0.8**	18.1±1.6	17.9±0.5	**0.0**	.15	**0.0**
pipesnt-pushstart(20)	8	8	6	7	8.0±0.0	8.8±1.3	**9.8±0.4**	9.7±0.5	8.5±0.5	**0.0**	0.1	**0.0**
pipesworld-pushend(20)	3	4	2	4	3.0±0.0	4.2±1.0	4.9±0.5	**5.2±1.2**	3.9±0.3	**0.0**	.09	**0.0**
scanalyzer-analyze(20)	9	9	3	9	**9.8±0.9**	9.4±0.5	9.2±0.4	7.3±1.0	9.1±0.3	.07	.37	.58
tpp-fuel(30)	8	**11**	7	**11**	7.5±0.5	**11.0±0.0**	**11.0±0.0**	**11.0±0.0**	8.1±0.3	**0.0**	1.0	**0.0**
woodworking-cut(20)	5	7	2	7	5.0±0.0	6.9±0.3	**9.2±0.9**	7.7±0.6	7.1±0.3	**0.0**	**0.0**	**0.0**
LMcut Total(1724)	814	844	654	837	814.0±2.3	856.9±8.5	**867.0±2.1**	838.7±4.9	824.7±2.1	**0.0**	**.01**	**0.0**
M&S IPC (1104)	479	**488**	451	481	478.8±0.4	484.8±0.4	484.0±0.0	481.4±1.4	486.4±0.8	**.01**	**.02**	**.01**
M&S Zerocost (620)	276	290	226	283	274.0±0.9	293.4±2.1	**310.2±2.1**	303.2±1.7	288.0±1.7	**.01**	**.01**	**.01**
M&S Total(1724)	755	778	677	764	752.8±0.7	778.2±1.9	**794.2±2.1**	784.6±2.1	774.4±1.2	**.01**	**.01**	**.01**

Table 1: Coverage comparison (# of instances solved in 5min, 2GB), **bold**=best. Zerocost domains are named as [original name]-[name of nonzero action]. Due to space, we only show the domains whose maximum pairwise coverage difference *MaxDiff* > 2. (We used the means of 10 runs for the randomized strategies.) Domains with *MaxDiff* ≤ 2 follows:
(1) *MaxDiff* $= 0$ (same coverages by all configuration and all runs): barman-opt11, floortile-opt11, grid, gripper, hanoi, parking-opt11, pegsol-opt11, psr-small, rovers, sokoban-opt11, tpp, transport-opt11, grid-fuel, gripper-move, parking-movecc, psr-small-open, zenotravel-fuel.
(2) $0 <$ *MaxDiff* ≤ 1: depot, driverlog, elevators-opt11, freecell, mystery, parcprinter-opt11, pathways, pipesworld-tankage, storage, tidybot-opt11, visitall-opt11, driverlog-fuel, floortile-ink, hiking-fuel, logistic00-fuel, nomystery-fuel, pathways-fuel, sokoban-pushgoal.
(3) $1 <$ *MaxDiff* ≤ 2: blocks, nomystery-opt11, pipesworld-notankage, zenotravel, depot-fuel, rovers-fuel, storage-lift, tidybot-motion.

fuel, elevator-up etc.), and assembly-type domains are modified to minimize resource usage (Woodworking-cut minimizes wood usage, etc.). We did not include domains with only a single action schema and standard domains which already had many zero-cost actions (these are already in the results for standard IPC domains). We refer to these 28 new domains as *zerocost domains*.

Figure 3 plots the size of the final plateau of the zerocost domain instances. As expected, many of these zerocost domains have large plateaus. Thus, in these cost-minimization problems, the search strategy within plateaus, i.e., tiebreaking, becomes very important.

6 Depth-Based Tiebreaking

In order to solve zerocost problems, the planner needs to perform an efficient knowledge-free search within a large, final plateau. One useful notion which can be used to both understand and control the search in this situation is the *depth* of a node, which represents the number of steps (edges in the search space graph) from the entrance of the plateau. Given a node n, if its current parent $parent(n)$ is from the other plateau, i.e., $parent(n)$ has a different f-value, or different h-value when the first tiebreaking is present, then $depth(n) = 0$. Nodes with $depth(n) = 0$ correspond to the *entrance* of the plateau. If n and $parent(n)$ are in the same plateau i.e. share the same f and h, $depth(n)$ is defined as $depth(parent(n)) + 1$. Based on this simple notion of depth, we propose three *depth-based tiebreaking* strategies, where the nodes are inserted into buckets associated with depths, and upon expansion, the buckets are chosen according to some policy. “First depth” (*fd*), “last depth” (*ld*), and “random depth” (*rd*) choose a bucket with the smallest depth, the largest depth, and a depth randomly selected at each expansion, respectively.

The effectiveness of each of these depth-based policies depends on the problem instance. Within the plateau region, all nodes have the same f and h values, and the goals can be near or far from the entrance. In the former case, the search should be focused around the entrance favoring the smaller depths (*fd*), and the behavior in the plateau should be much like breadth-first. In the latter case, the planner should greedily explore the various area of the plateau by preferring largest depth (*ld*), much like in depth-first. It may also be possible for a goal to be at an intermediate depth, in which case *fd* could take too much time to reach that depth, and *ld* may greedily pass and miss that depth. By an adversary argument, *rd*, which selects a random depth and has no depth bias would seem to be the safest policy.

Depth-based tiebreaking has no effect when used as a second-level tirebreaking policy with a domain with positive costs only *and* h-based first-level tiebreaking policy. This is because most actions result in an updated h-value, so almost all nodes have depth 0.

Tiebreaking within Depth Buckets Since there can be multiple nodes within the same depth bucket, a further tiebreaking criterion may be necessary to break ties among them. We could, for example, apply *lifo* or *fifo* policies at this level – note that $[h, fd, fifo]$ and $[h, ld, lifo]$ are equivalent to $[h, fifo]$ and $[h, lifo]$, respectively.

However we use a Random Order (*ro*) policy, which randomly selects an element from the depth bucket selected by the depth-based tiebreaking. This is because the effectiveness of the tiebreaking behavior within a bucket can be affected by accidental biases, e.g., names/orders of action schema in the PDDL domain definition (Vallati et al. 2015). Thus, we avoid bias at this level of tiebreaking by using *ro* and assess its expected/average performance.

6.1 Evaluating Depth-Based Tiebreaking

We evaluated three 3-level tiebreaking strategies. In addition to the 35 IPC benchmark domains with 1104 instances used in the previous set of experiments, we used 28 zerocost domains with 620 instances. For randomized strategies, we show the coverage (mean $\pm$ sd) on 10 independent runs.

We compared $[h, fd, ro]$, $[h, ld, ro]$, and $[h, rd, ro]$. These all use h as the first-level tiebreaking criterion, one of *fd*, *ld*, *rd* as the depth-based 2nd-level tiebreaking criterion, and finally, *ro* as the 3rd-level criterion. Since these configurations are randomized, we run each configuration with 10 different random seeds. To see whether differences among the mean coverages were statistically significant, we applied the non-parametric Wilcoxon's signed-rank test.

Table 1 shows the coverage (mean $\pm$ sd), along with the rightmost columns showing the Wilcoxon test p values for $[h, rd, ro]$ vs. 3 other strategies. In many domains, the performance was significantly affected by 2nd-level tiebreaking, and $[h, rd, ro]$ dominated the others. Although $[h, ld, ro]$ and $[h, rd, ro]$ performed similarly on most domains, the performance of $[h, rd, ro]$ in some domains are notable (e.g., Cybersec, Woodworking-cut). The standard deviation of $[h, rd, ro]$ coverage tends to be smaller than that of $[h, ld, ro]$, indicating that $[h, rd, ro]$ is robust with respect to random seeds. $[h, fd, ro]$ is mostly dominated by $[h, rd, ro]$ and $[h, ld, ro]$, except in Scanalyzer-analyze.

Depth-Based Tiebreaking Without Considering h In Sec. 4, we showed that $[lifo]$ tiebreaking (without considering h) is sufficient for the standard IPC benchmarks – the performance of $[lifo]$, $[h, lifo]$, and $[h, fifo]$ are comparable. Table 1 shows that $[rd, ro]$, which randomly selects an element from a randomly selected depth-bucket, dominates $[h, fifo]$, and performs comparably to $[h, lifo]$. Although $[rd, ro]$ behaves in a less greedy/depth-first manner than $[lifo]$, it explores nodes with high depth sufficiently often so that even if *lifo* behavior (seeking nodes that are far from the plateau entrance) is required, $[rd, ro]$ will eventually find the solution. Moreover, there are some domains (pipesworld-pushend and woodworking-opt11) where a the more randomized behavior of $[rd, ro]$ is advantageous. Thus, overall, $[rd, ro]$ performs moderately well, and *neither h nor lifo-behavior is necessary in order to obtain performance that is competitive with the standard tiebreaking strategies.*

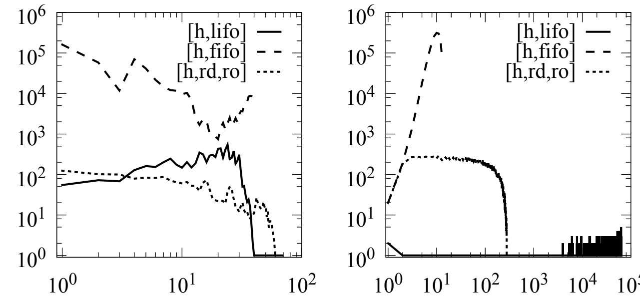

Figure 5: Number of nodes (y-axis) expanded per depth (x-axis) in the final plateau for Openstacks p10 (left) and Woodworking-cut p04 (right) with different tiebreakings.

Is Depth-Based Tiebreaking Necessary? We have shown that $[h, rd, ro]$ performs well overall, but one might wonder whether the power of this strategy really comes from depth-based tiebreaking, or from randomness. Table 1 shows that $[h, ro]$ performs poorly, so clearly, random tiebreaking combined with h-based tiebreaking is not sufficient. The reason that $[h, ro]$ performs so poorly is that if we select uniformly from the bucket of all open nodes in the plateau, there is a very strong bias for selecting a node with low depth, simply because at any given point during the search in the plateau region, more nodes closer to the plateau entrance (i.e., lower depth) will have been generated. By randomly selecting a depth bucket, $[h, rd, ro]$ explicitly eliminates this bias for selecting nodes for low depth.

Search Behavior Within a Plateau To understand the behavior of depth-based policies, we plotted the histogram of the depths of search nodes opened by the most successful depth-based strategy, $[h, rd, ro]$, as well as the standard $[h, fifo]$, $[h, lifo]$ strategies in the final plateau, plateau$(f^*, 0)$ until the solution is found. Although $[h, fifo]$ and $[h, lifo]$ do not operate with an explicit notion of "depth", they are equivalent to $[h, fd, fifo]$ and $[h, ld, lifo]$, respectively, so we recorded and plotted the depths according to $[h, fd, fifo]$ and $[h, ld, lifo]$.

Figure 5 shows the result on Openstacks-opt11 p10 (left) and Woodworking-cut p04 (right). In both instances, we observed that the depth-first behavior of $[h, lifo]$ results in deeper search, missing the key branch at intermediate depths. On the other hand, the breadth-first behavior of $[h, fifo]$ often gets stuck spending an excessive amount of time searching around the plateau entrance. $[h, rd, ro]$ is balancing the search at various depths, which results in successfully solving more problems within the time limit (Table 1).

Comparison With ε-Cost Transformation An alternative approach to addressing the large plateaus in zero-cost

Domain	$[h, \mathit{fifo}]/\varepsilon$	$[h, \mathit{lifo}]/\varepsilon$	$[h, \mathit{fd}, \mathit{ro}]$	$[h, \mathit{rd}, \mathit{ro}]$
LMcut Zerocost	261	259	257.4±2.0	**294.2±2.3**
M&S Zerocost	282	282	274.0±0.9	**310.2±2.1**

Table 2: Comparison of depth-based tiebreaking methods vs. standard $[h, \mathit{fifo}]$ and $[h, lifo]$ methods applied to ε-cost-transformed versions of the problem instances

domains is to eliminate plateaus by introducing artificial gradients in the search space. For example, the cost of all zero-cost actions can be replaced by a small $\varepsilon \ll 1$, where ε is chosen such that the optimal cost for the result of this *ε-cost transformation* ("ε-transformation") is the same as the cost of the optimal solution to the original domain with zero costs when the ε-transformed costs are mapped back to 0.

We evaluated the $[h, \mathit{fifo}]/\varepsilon$ and $[h, \mathit{lifo}]/\varepsilon$ strategies, which are the standard $[h, \mathit{fifo}]$ and $[h, \mathit{lifo}]$ tiebreaking strategies applied to the ε-transformed version of the problems. Since Fast Downward only supports integer costs, we implemented/simulated the transformation by multiplying the non-zero costs by 10^6, and assigning cost 1 to zero-cost actions – in effect, $\varepsilon = 10^{-6}$. Table 2 shows that $[h, \mathit{fifo}]$ and $[h, \mathit{lifo}]$ with ε-transformation perform comparably to $[h, \mathit{fd}, \mathit{ro}]$, but are outperformed by $[h, \mathit{rd}, \mathit{ro}]$. The similarity in performance between ε-transformation and $[h, \mathit{fd}, \mathit{ro}]$ can be explained by the fact that OPEN is sorted according to $f(n)+k(n)\varepsilon$, where $k(n)$ is a number of zero-cost actions in the path to node n, while expansion order of FirstDepth is equivalent to $f(n) + \mathit{depth}\,(n)\,\varepsilon$. ($\mathit{depth}\,(n) \le k(n)$ because $k(n)$ accounts zero-cost actions also in non-final plateaus). One advantage of the ε-transformation is that it can be implemented by transforming the input problem and does not require implementation of depth-based buckets in the search algorithm. On the other hand, there are two issues with the ε-transformation: (1) ε must be chosen carefully – admissibility is lost when $k(n)\varepsilon \approx 1$, and (2) the number of possible g and f values becomes very large, making it difficult to use efficient $O(1)$ array-based implementation of the OPEN list and requiring the use of a heap-based $O(\log n)$ OPEN list.

7 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 tested the robustness of the standard $[h, \mathit{lifo}]$ and $[h, \mathit{fifo}]$ strategies, as well as $[h, \mathit{rd}, \mathit{ro}]$, 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 (recall that robustness wrto random seed was shown in Sec. 6.1.)

Domain	$[h, \mathit{fifo}]$	$[h, \mathit{lifo}]$	$[h, \mathit{rd}, \mathit{ro}]$ (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 3: Total coverages of $[h, \mathit{fifo}]$, $[h, \mathit{lifo}]$ and $[h, \mathit{rd}, \mathit{ro}]$ (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 $[h, \mathit{rd}, \mathit{ro}]$ to constantly perform better than the traditional tiebreaking methods.

The results are shown in Table 3 (We also included the original 10 runs from Table 1). We statistically analyzed the results for $[h, \mathit{rd}, \mathit{ro}]$ 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.

8 Related Work

Previous work on escaping search space plateaus has focused on non-admissible search. DBFS (Imai and Kishimoto 2011) adds stochastic backtracking to Greedy Best First Search (GBFS) to avoid being misdirected by the heuristic function. Type based bucket (Xie et al. 2014) classifies the plateau of GBFS according to the $[g, h]$ pair and distributes the effort. Marvin (Coles and Smith 2007) learns plateau-escaping macros from the Enhanced Hill Climbing phase of the FF planner (Hoffmann and Nebel 2001), and the use of these macros is inadmissible. Hoffmann gives a detailed analysis of the structure of the search spaces of satisficing planning (2005; 2011). (Benton et al. 2010) proposes inadmissible technique for temporal planning where short actions do not increase makespan. (Cushing, Benton, and Kambhampati 2010) investigates "ε-cost traps"($\varepsilon = \frac{\min cost}{\max cost}$), showing that (non-admissibly) treating all actions as unit cost sometimes finds an optimal plan quickly. (Wilt and Ruml 2011) also analyzes inadmissible distance-to-go estimates. To our knowledge, plateaus have not been previously investigated for cost-optimal planning with admissible search. Admissible and inadmissible search differ significantly in how non-final plateaus (plateaus with $f < f^*$) are treated: Inadmissible search can skip or escape plateaus whenever possible, while admissible search cannot, unless it is the final plateau ($f = f^*$, $h = 0$) and a solution is found.

The PLUSONE cost-type (or distance-to-go) is a non-admissible search technique in the Fast Downward/LAMA planner (Richter and Westphal 2010) which increases all action costs by 1. This technique explicitly targeted zero-cost actions, and resulted in significantly better performance in the IPC-6 satisficing track (Richter and Westphal 2010, p.137, Sec. 3.3.2). Unlike PLUSONE, depth-based tiebreaking is admissible. Also, unlike PLUSONE, depth-based tiebreaking does not necessarily favor smaller depth over larger depth. LAMA prefers smaller cost (including the increased cost), which biases the search toward nodes with fewer zero-cost actions on their path. This bias is similar to the $[h, fd, ro]$ policy, the worst performer among all depth-variants in our experiments (Table 1). The best depth-based methods are $[h, ld, ro]$ and $[h, rd, ro]$, which do not prefer smaller depth.

9 Conclusion

In this paper, we evaluated standard tiebreaking strategies for A^*. We showed that contrary to conventional wisdom, tiebreaking based on the heuristic value is not necessary to achieve good performance, and proposed a new framework for defining tiebreaking policies based on *depth*. We showed that a depth-based, randomized strategy $[h, rd, ro]$, which uses the heuristic value, but explicitly avoids depth and ordering biases present in previous methods, significantly outperforms previous strategies on domains with zero-cost actions, including practical application domains with resource optimization objectives in the IPC benchmarks. The proposed approach is highly effective on domains where zero-cost actions create large plateau regions where all nodes have the same f and h costs and the heuristic function provides no useful guidance.

References

Benton, J.; Talamadupula, K.; Eyerich, P.; Mattmüller, R.; and Kambhampati, S. 2010. G-Value Plateaus: A Challenge for Planning. In *Proc. ICAPS*, 259–262.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of Action Generation for Cyber Security Using Classical Planning. In *Proc. ICAPS*, 12–21.

Burns, E. A.; Hatem, M.; Leighton, M. J.; and Ruml, W. 2012. Implementing Fast Heuristic Search Code. In *Proc. Symposium on Combinatorial Search*.

Coles, A., and 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.; and Kambhampati, S. 2010. Cost Based Search Considered Harmful. In *Proc. Symposium on Combinatorial Search*.

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

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

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

Helmert, M., and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proc. ICAPS*.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *J. ACM* 61(3):16:1–16:63.

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

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

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.

Holte, R. C. 2010. Common Misconceptions Concerning Heuristic Search. In *Proc. Symposium on Combinatorial Search*.

Imai, T., and Kishimoto, A. 2011. A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning. In *Proc. Symposium on Combinatorial Search*.

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

Nilsson, N. 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.

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

Röger, G., and Helmert, M. 2010. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In *Proc. ICAPS*, 246–249.

Vallati, M.; Hutter, F.; Chrpa, L.; and McCluskey, T. L. 2015. On the Effective Configuration of Planning Domain Models. In *Proc. IJCAI*.

Wilt, C. M., and Ruml, W. 2011. Cost-Based Heuristic Search is Sensitive to the Ratio of Operator Costs. In *Proc. Symposium on Combinatorial Search*.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-Based Exploration with Multiple Search Queues for Satisficing Planning. In *Proc. AAAI*, 2395–2402.