Improving Greedy Best-First Search by Removing Unintended Search Bias

Submision 1537

Abstract

Recent enhancements to greedy best-first search (GBFS) such as DBFS, ϵ-GBFS, Type-GBFS) improve performance by occasionally adopting a non-greedy node expansion policy, resulting in more exploratory behavior. However, previous exploratory mechanisms do not address exploration within the space sharing the same heuristic estimate (plateau). In this paper, we show these two modes of exploration (diversification), which work *inter*-(across) and *intra*-(within) plateau, are orthogonal. We also introduce IP-diversification, a method combining Minimum Spanning Tree and randomization, which addresses "breadth"-bias instead of the "depth"-bias addressed by the existing methods. We evaluate IP-diversification for both intra- and inter-plateau diversification, and show that it significantly improves performance in several domains. Finally, we show that combining diversification methods results in a planner which improves upon the state-of-the-art for satisficing planning.

1 Introduction

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 $f(n)$, the expansion priority of node n is based only on a heuristic estimate of the node, i.e., $f(n) = h(n)$ (in contrast with A^* search, where the node priority also considers $g(n)$, the cost from the start state to n, and $f(n) = g(n) + h(h)$). Although GBFS ignores solution optimality, it 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 and 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.

The problem is exacerbated by the fact that GBFS tends to be combined with inadmissible heuristic functions such as the FF heuristic (Hoffmann and Nebel 2001), Causal Graph (Helmert 2006) or Landmark-count (Richter, Helmert, and Westphal 2008) heuristics. An inadmissible heuristic can cause nodes which are close to the goal to be incorrectly labeled as unpromising, causing GBFS to delay expanding them until all other nodes in the current local minima with smaller h-values have been expanded.

Recently, several approaches have been proposed for alleviating this problem, e.g., DBFS (Imai and Kishimoto 2011), ϵ-GBFS (Valenzano et al. 2014) and Type-GBFS (Xie et al. 2014). They improve the search performance by occasionally expanding nodes which do not have the lowest h-value, i.e., diversifying the search. These diversified algorithms provides an opportunity to expand nodes that are mistakenly overlooked due to errors made by the heuristic functions. A common objective among these methods is the removal of some (undesirable/unintended) *bias*, thereby encouraging *exploration* by the search process and adding *diversity* in decision making process. In this paper, we use the terms "exploration", "diversity", and "bias removal" interchangeably. Existing methods for diversification have two issues: First, previous methods all employ h-based diversification as part of their algorithms in order to avoid the bias toward the nodes with smaller estimates. However, h-based diversification cannot detect the bias *among nodes with the same h-cost*. Second, as we see later, they 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 paper, we first show that a recently proposed depth-based tie-breaking strategy for A^* also improves the performance of GBFS by diversifying the depth within each h-plateau. We then show that this depth diversification and Type-GBFS are instances of a *bucket*-based diversification strategy: Depth diversification applies bucket diversification within a plateau, and Type-GBFS applies it between plateaus. We compare their empirical performance and show that their improvements are orthogonal – this effectively shows that *inter*-plateau and *intra*-plateau diversification are two orthogonal usages of diversification. 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 diversification. Finally, we show that by combining several intra/inter plateau diversification strategies, we can improve upon state-

of-the-art planners (LAMA, the IPC2011 winner, and Type-LAMA, the non-portfolio planner with the highest coverage in IPC2014) in terms of coverage.

2 Preliminaries and Background

We first define some notation and the terminology used throughout the rest of the paper. $h(n)$ denotes the estimate of the cost from the current node n to the nearest goal node. $g(n)$ is the current shortest path cost from the initial node to the current node. $f(n) = g(n) + h(n)$ is the estimate of the resulting cost of the path to a goal containing the current node. We omit the argument (n) unless necessary.

A *sorting strategy* for a best first search algorithm tries to select a single node from the open list (OPEN). Each sorting strategy is denoted as a vector of several *sorting criteria*, such as [criterion$_1$, criterion$_2$, . . ., criterion$_k$], which means: First, select a set of nodes from OPEN using criterion$_1$. 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.

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

A sorting strategy fails to select a single node when some nodes share the same sorting keys. In such cases, a search algorithm must select a node according to a *default* tie-breaking criterion, criterion$_k$, such as *fifo* (first-in-first-out), *lifo* (last-in-first-out) or *ro* (random ordering). For example, an A^* using h and *fifo* tie-breaking is denoted as $[f, h, \textit{fifo}]$. By definition, default criteria are guaranteed to return a single node from a set of nodes. When the default criterion does not matter, we may use a wildcard $*$ as in $[f, h, *]$.

Given a sorting strategy, a *plateau* is a set of nodes in OPEN whose elements are indistinguishable according to the criteria (up to the default criterion). In the case of A^* using tie-breaking with h (i.e. $[f, h]$), this is the set of nodes with the same f cost and the same h cost.

Finally, OPEN list *alternation* (Röger and 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.

Depth-Based Tie-breaking To date, there has been relatively little work on tie-breaking policies for BFS. Recently, Asai and Fukunaga (2016) performed an in-depth investigation of tie-breaking strategies for A^*. In A^*, the tie-breaking policy was found to have a significant effect on the performance when the search plateau is huge. While using h value as the first tie-breaking criterion is a common strategy that is useful in many cases, it loses its effectiveness when the domain contains many zero-cost actions because there are large number of nodes with $h = 0$, preventing h-based tie-breaking from providing a useful gradient toward goals. In the most commonly used sorting strategies, $[f, h, \textit{fifo}]$ or $[f, h, \textit{lifo}]$, the search has a strong bias to focus on either the regions of smaller (*fifo*) or larger (*lifo*) search depth of the plateau, which may cause failure to find the solution within a given time limit.

To address the issue caused by the search bias within a plateau, they proposed a notion of *depth* and diversified the search over different depths within a plateau. A depth is an integer representing the step-wise distance from the *entrance* of the plateau (the most recent state which entered the plateau, along the path from the initial state). The depth $d(s)$ of a state s is 0 when s and the parent state t has the piecewise-different (f, h) pair, and $d(s) = d(t) + 1$ when they are same (i.e. $f(s) = f(t) \wedge h(s) = h(t)$). The former case applies when the child is an entrance, while the latter case applies when the parent and child are in the same plateau. The nodes are stored in buckets indexed by depth, and expansions are allocated evenly across different buckets. The resulting sorting strategy is denoted as $[f, h, \langle d \rangle, *]$.

For GBFS, to our knowledge, there are 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. As a consequence, improvements to GBFS have been primarily achieved by addressing other aspects, such as modifying the evaluation scheme (Richter and Westphal 2010, lazy evaluation), queue alternation (multiple heuristic functions), preferred operators (Hoffmann and Nebel 2001), and diversification.

Diversified Search Algorithms One recent class of improvements to GBFS seeks to introduce exploration (diversity) to the search process, as exemplified by DBFS (Imai and 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. In GBFS, a node will not be expanded until it expands all nodes with a lower h-value in the current local minima. Thus, search progress can be delayed when either a good node is mistakenly assigned a poor h-value, or bad nodes are assigned promising h-values. Diversifcation-based algorithms allow the search to escape the local minima by relaxing the (h-based) best-first node expansion order.

KBFS(k) (Felner, Kraus, and 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, weighted alternating OPEN list using $[h, *]$ and $[ro]$ (no sorting criteria): *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 no-

tation $\langle \ldots \rangle$, 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*($[h, *], [\langle g, h \rangle, ro]$).

3 Intra- and Inter-plateau Diversification

Previous diversification techniques primarily address the problem of heuristic errors, where promising nodes are incorrectly labeled as unpromising, or vice versa. They introduce diversification by occasionally expanding nodes with higher estimates, both in order to avoid expanding more dead-end states labeled as promising, or to find a good state hidden behind nodes mislabeled as unpromising. Since this type of diversification operates across different search plateaus, we call them *inter-plateau* diversification.

However, there is another type of diversification which we call *intra-plateau* diversification, which works within a particular plateau: The expansion order is changed only among the nodes within a plateau.

*Unlike inter-plateau diversification, which addresses **incorrect** information, intra-plateau diversification addresses the problem of **insufficient** information.* Since we do not know *a priori* which nodes in a plateau are better other nodes in the same plateau, by an adversarial argument, the safest strategy is to avoid biased choices – in the absences of useful heuristic knowledge which differentiates among a set of nodes, an expansion policy which is biased to expand some particular group of states within a plateau can be exploited by an adversary which seeks to hide a solution.

One example of insufficient heuristic information is the case where there are many zero-cost edges in the search graph, and the heuristics fail to differentiate among the nodes with the same h-value. This issue is addressed by depth-based diversification for A^* (Asai and Fukunaga 2016), which is a type of intra-plateau diversification.

Another type of lack of information occurs when relaxation or abstraction makes the heuristics uninformative. Consider an extreme case when the heuristic function returns a constant value c for all nodes. Since there is only a single plateau in the search space, h-based inter-plateau diversification does not affect the expansion order at all. This clearly indicates a limitation of inter-plateau diversification.

Intra-plateau diversification is different from tie-breaking because tie-breaking implies choosing a *better* node based on some additional criteria, implicitly sorting the nodes. Diversification, in contrast, allocates search effort equally among groups of nodes.

Bucket-Based Diversification The notions of inter-vs-intra plateau diversification allows us to discuss and compare depth diversification (Asai and Fukunaga 2016) and Type-GBFS (Xie et al. 2014) within a unified framework – it turns out that these are in fact essentially the same algorithm, except that they are using different key values (metrics) in different contexts (inter-vs-intra plateau).

Definition 1 (Bucket-based diversification)**.** *Bucket-based diversification is a general strategy which categorize the search nodes according to a set of distance metrics. Search effort is allocated evenly among buckets, removing the cardinality bias among buckets.*

Depth diversification originally addressed the issue of zero-cost actions in the optimal planning with A^*. The resulting configuration is $[h, \langle d \rangle]$.

Observation 1. *Depth diversification, $[h, \langle d \rangle]$, is an instance of bucket-based diversification applied to intra-plateau diversification using depth d as a metric.*

Observation 2. *Type-GBFS with $\langle h \rangle$ is an instance of bucket-based diversification applied to inter-plateau diversification using heuristic function h as a metric.*

The above Type-GBFS configuration, $\langle h \rangle$, differs from the $\langle g, h \rangle$ type-bucket which was the best configuration in (Xie et al. 2014). This is because the $\langle g, h \rangle$ configuration (also included in our evaluation below) has not only the inter-plateau effect (due to h) but also some intra-plateau effect due to g. Since d represents the path length from plateau entrance while g represents the path length from the initial state, g includes d. Thus, diversification based on g also affects the expansion distribution of depth d. Note that, as with GBFS in state-of-the-art satisficing planners, all configurations are evaluated under a unit cost transformation where all actions costs are transformed to 1.

Empirical Comparison 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 diversification. In the following experiments, we empirically show that they achieve orthogonal performance improvements. This indicates that inter/intra-plateau diversification in fact addresses orthogonal issues of *incorrect* and *insufficient* information, respectively.

Throughout this paper, experiments are conducted on Amazon EC2 c4.8xlarge instances (Xeon E5-2666 @ 2.9GHz w/o HyperThreading, TurboBoost) under singe-user tenancy. We used 4GB memory limit and 5 minutes time limit, on IPC 2011 and 2014 instances. Following previous work (Valenzano et al. 2014; Xie et al. 2014), all configurations are evaluated under unit cost transformation because we are solely focused on the coverage and the first solution.

The "*marked" results in Table 1 **(1)** shows that both intra- (**hd**) and inter-plateau (**ht(h)**) diversification improve the performance while the domains they improved are complementary. $\langle g, h \rangle$ (**ht(g,h)**) has intermediate results because it reflects both the effects of full inter-plateau diversification as well as partial intra-plateau diversification. Unlike depth d, g is not fully targeted toward diversity within h-plateau. We also see that the improvement is affected by the heuristics being used. For example, in parking, intra-plateau diversification doubled the coverage with h^{FF} while it has only a negative effect in h^{CG}. (**Since hd and ht(h)/ht(g,h) are complementary, combining these algorithms also results in overall improvements, see Supplemental Table S8.**)

Based on these results, we conclude that (1) inter-plateau diversification and intra-plateau diversification are orthogonal and complementary; and (2) the benefits from different type of diversification (inter/intra-plateau) depend on the problem domain and heuristic used.

	(1) Intra- vs. Inter-plateau diversification (4 runs average)								**(2)** IP diversification (4 runs)				**(3)**State-of-the-Art results w/ LAMA (10 runs)		
	h^{FF}				h^{CG}				h^{FF}		h^{CG}				
Diversification	none	intra	inter	mixed	none	intra	inter	mixed	intra	inter	intra	inter	none	inter	both
Algorithm	**h**	**hd**	**ht(h)**	**ht(g,h)**	**h**	**hd**	**ht(h)**	**ht(g,h)**	**hb**	**hB**	**hb**	**hB**	2011	Type	Btdb
Total	229.8	264.3	232	250.3	227.5	242.3	229.8	246.3	261.5	**296.6**	235	**263.3**	386.1	387.3	**399.6**
barman11	8.0	8.3	8.0	*10.5	0.0	0.0	0.0	0.0	8.3	**16.8**	0.0	0.0	**19.0**	16.4	17.9
elevators11	18.5	14.0	15.3	16.0	9.0	8.0	9.0	8.8	18.0	**19.0**	9.8	**13.3**	*20.0*	19.0	19.5
floortile11	5.8	6.3	***7.3**	***7.3**	0.0	0.0	*1.8	***2.3**	4.3	5.3	0.0	0.5	3.0	**5.6**	4.8
nomystery11	9.0	7.0	9.3	***16.5**	7.0	6.0	8.0	***15.8**	6.8	7.5	6.3	5.5	10.0	**17.4**	15.7
openstacks11	11.0	*18.3	*13.0	11.0	10.0	***14.8**	10.0	10.0	**19.5**	17.3	12.0	13.5	20.0	20.0	20.0
parcprinter11	20.0	20.0	20.0	20.0	**20.0**	**20.0**	18.3	19.5	20.0	20.0	19.8	13.0	20.0	19.8	19.4
parking11	10.5	***20.0**	9.3	9.8	18.0	18.0	14.8	11.5	17.0	13.3	**19.8**	**19.8**	20.0	20.0	20.0
pegsol11	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	19.8	20.0	20.0	20.0
scanalyzer11	15.5	15.3	*18.3	*17.8	20.0	20.0	19.8	20.0	16.8	18.8	20.0	20.0	19.0	19.1	*19.4*
sokoban11	19.0	19.0	17.8	17.3	16.0	16.0	17.0	16.8	18.5	18.3	15.8	16.0	*17.0*	16.9	16.8
tidybot11	16.0	16.0	14.8	16.0	16.0	*18.0	***18.3**	***18.3**	14.8	**16.3**	18.0	16.8	16.0	16.0	*16.5*
transport11	0.0	0.0	0.0	0.0	10.0	11.5	8.5	9.0	0.0	0.0	11.0	**12.3**	*11.0*	*11.0*	10.9
visitall11	3.0	3.0	*6.8	*6.0	3.0	3.0	*5.5	*6.3	5.3	**11.3**	3.8	**10.0**	20.0	20.0	20.0
woodworking11	2.0	2.0	*5.0	*3.5	2.0	2.0	*5.3	2.8	1.5	**14.8**	2.0	**14.8**	20.0	20.0	20.0
barman14	0.0	0.0	0.0	*1.0	0.0	0.0	0.0	0.0	0.0	**7.8**	0.0	0.0	**15.0**	9.5	13.1
cavediving14	7.0	7.0	7.3	7.0	7.0	7.0	7.3	7.0	7.0	7.0	7.3	7.0	7.0	*7.1*	7.0
childsnack14	0.0	***4.0**	0.0	0.0	1.0	***6.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	*0.3*
citycar14	0.0	0.0	*1.8	***7.0**	0.0	0.0	0.0	***8.3**	0.0	2.8	0.3	1.3	2.0	**5.5**	4.1
floortile14	2.0	2.0	2.0	2.0	0.0	0.0	*1.8	***2.0**	2.0	2.3	0.0	0.3	2.0	*2.1*	2.0
ged14	19.0	19.0	15.0	13.8	0.0	0.0	*6.8	***9.5**	18.8	12.5	0.0	4.3	20.0	20.0	20.0
hiking14	20.0	20.0	19.5	19.8	18.0	16.8	18.3	***19.5**	18.0	20.0	15.8	18.3	18.5	17.5	*19.0*
maintenance14	11.0	8.0	8.3	10.8	16.0	16.0	15.8	**16.3**	6.8	10.8	14.0	15.8	1.0	**5.5**	4.7
openstacks14	0.0	*12.3	0.3	0.0	0.0	***3.5**	0.0	0.0	**14.8**	11.3	0.0	2.3	20.0	20.0	20.0
parking14	3.3	*7.3	2.0	1.3	6.8	*9.8	2.0	0.8	5.8	1.5	**11.3**	8.3	**19.1**	16.7	19.0
tetris14	1.3	*6.8	1.3	*3.3	17.8	18.0	12.5	14.0	**9.3**	8.3	**20.0**	18.0	9.3	7.3	**13.9**
thoughtful14	8.0	9.0	*10.3	***13.0**	5.0	5.0	5.0	5.0	8.8	11.0	4.5	5.0	14.0	**15.0**	14.2
transport14	0.0	0.0	0.0	0.0	5.0	3.0	4.5	3.3	0.0	0.0	4.0	**6.0**	3.3	2.7	*3.5*
visitall14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**3.3**	0.0	**2.0**	**20.0**	16.9	18.0

Table 1: Number of solved instances (5 min, 4Gb RAM). **h**: standard GBFS $[h]$. **hd**: intra-plateau bucket diversification $[h, \langle d \rangle]$, **ht(h)**: inter-plateau bucket diversification (i.e. Type-GBFS) $alt([h], [\langle h \rangle, ro])$, **ht(g,h)**: Best Type-GBFS by (Xie et al. 2014), $alt([h], [\langle g, h \rangle, ro])$, **hb**: intra-plateau IP diversification $[h, r_{\text{BIP}}]$, **hB**: inter-plateau IP diversification $alt([h], [r_{\text{BIP}}])$. Unless specified, all configurations use *fifo* tiebreaking (FastDownward default) – (**see supplements for *lifo* and *ro***). **(1)** compares Intra- vs. Inter-plateau diversification. From the "*marked" per-domain results which exceed pure GBFS **h** by > 1.5, we see that the effect of intra-plateau (hd) and inter-plateau (ht) are complementary. **(2)** shows the results of IP diversification. For each heuristic, we **highlighted** the best results among (h,hd,ht,hb,hB) when $(\max - \min) \geq 2$. (hd vs. hb) and (ht vs. hB) both show that bucket-based (depth) and IP-based (breadth) diversifications are orthogonal. Finally, **(3)** shows adding **d,b,B** to Type-LAMA (=LAMA/2011 + **t**) improves upon the state-of-the-art. We also *highlighted* the best results when $(\max - \min) < 2$.

4 Breadth-Based Diversification: Invasion Percolation

One problem with bucket-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 bucket. This makes bucket-based diversification 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 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 bucket-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 branch before expanding L_1 and low-b branch. *fifo* performs Breadth-First Search (BrFS), 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*/BrFS, but 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 bucket-diversification addresses the depth bias of BrFS. However, even though it distributes the effort among various depths, the probability of

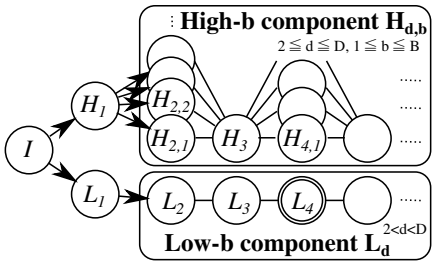

Figure 1: Example case exhibiting large bias in the branching factor depending on the subgraph.

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 *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), and the combination is called *Invasion Percolation* (Wilkinson and Willemsen 1983).

Invasion Percolation Invasion Percolation 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 never changes, and the initial nodes are marked by default. In each iteration, it marks a neighbor of marked nodes with the least edge value leading to it. Marked nodes represent the porous sites whose air is replaced by the water (invader). Barabási (1996) showed that this BIP algorithm is equivalent to applying Prim's method for MST (Prim 1957) on a randomly weighted graph. Since Prim's method finds a MST minimizing the edge cost sum, BIP simulates the water greedily trying to expand with the least friction pathways.

Figure 2 shows an example of a 2-dimensional lattice after running this algorithm for a certain length of time. The initial nodes are on the leftmost edge of the rectangular region, i.e. the fluid percolates from the left. The resulting structure has holes of various size 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. This behavior is critical to forming a fractal structure.

Invasion Percolation for Search Diversification We apply the BIP model for diversification of best-first search. Since the previous work on BIP was done on physical simulations with relatively small graphs, we believe this is the first attempt to apply BIP to complex implicit graphs.

Consider applying BIP to the DAG in Figure 1. Note that there is a non-negligible probability that the search finds the solution without expanding high-b branch: This occurs when the value $v(H_1)$ of H_1 is higher than the value of

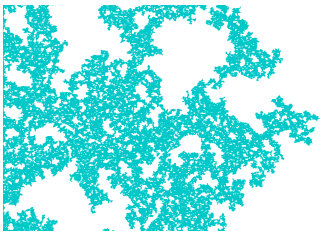

Figure 2: Invasion Percolation on 2-dimensional lattice (Monnerot-Dumaine 2006)

any of $L_1 \ldots L_4$, whose probability is 1/5 (follows from $\int_0^1 dy Pr(\forall i; v(L_i) \leq y) = \int_0^1 y^4 dy$). 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).

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 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, BIP tends to start looking at the other branch. Since this is affected by the least width of a particular subgraph rather than the maximum, it is less likely to suffer from the kind of pathological behavior exemplified by Figure 1.

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 diversification, 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, this follows Prim's method. For inter-plateau diversification, we alternate the expansion between standard GBFS and a queue sorted by r_{BIP}: *alt*($[h]$, $[r_{\text{BIP}}]$), just as in Type-GBFS.

Node expansion order according to r_{BIP} differs significantly from that of *ro* (pure random selection). (**See Supplemental Table S1, Table S2 for an empirical comparison**.) *ro* is equivalent to shuffling all nodes at every single node expansion, 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 2. Since all nodes are assigned a new random value at each expansion, the nodes inside the holes are more likely to be expanded and the holes, if any, will tend to be filled up more quickly. Thus, running *ro* would result 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 our BIP-based method for search diversification. The search spaces of the domains we are interested in, e.g., 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 and Liu 1965; Edmonds 1967; Tarjan 1977; Gabow et al. 1986) to search diversification is a direction for future work.

5 Performance Evaluation

Evaluation under Eager GBFS In this section, we evaluate the effects of bucket-based diversification and IP-diversification on Eager GBFS, where nodes are evaluated as soon as they are generated. Since the effect of diversification could be heuristic-dependent, we tested h^{FF} and h^{CG}.

We compared the performance of (h) the standard GBFS $[h]$ and its enhancements: aforementioned (hd) and (ht) as well as (hb) intra-plateau IP diversification $[h, r_{\text{BIP}}]$ and (hB) inter-plateau IP diversification $alt([h], [r_{\text{BIP}}])$. All configurations uses *fifo* default tie-breaking which is the default configuration in Fast Downward, except Type-bucket in Type-GBFS (which uses *ro*, as specified in their paper). (**See Supplemental Table S2 for comparisons to the other default tie-breaking methods (*lifo* and *ro*). Table S5 also contains Lazy-GBFS results, where nodes are evaluated when they are expanded**.)

Results are shown in Table 1 **(1,2)**. Overall, both bucket- and IP-diversification, applied to both intra- and inter-plateau, offer improvements to both heuristics. None of (hd,hb,ht,hB) dominate each other on a per-domain basis. For example, with h^{CG}, (hb) outperforms (hd) on parking14 and tetris14, while openstacks14 and childsnack14 are solved only by (hd) and other configurations have 0 coverages. Similarly with h^{FF}, while (hb) offer improvement in visitall11 ($3.0 \rightarrow 5.3$), (hd) does not offer any improvements, and vice versa in childsnack14 ($0.0 \rightarrow 4.0$ by (hd)). Regarding inter-plateau behavior, (hB) provides huge improvements to h^{FF} on barman11/14 compared to (ht), while (ht) significantly outperforms (hB) on floortile11/14, nomystery11, citycar14 with both h^{FF} and h^{CG}.

These results indicate that the behavior of bucket and IP diversification are orthogonal for both intra-plateau and inter-plateau diversification – bucket and IP diversification address different diversity criteria (depth vs breadth). Nevertheless, (hB) outperforms other algorithms in IPC2011 and is competitive with (hd) and (ht(g,h)) in IPC2014, resulting in the highest overall coverage in both heuristics.

State-of-the-Art Performance Finally, we apply these techniques to LAMA (Richter and Westphal 2010), the winner of the IPC2011 satisficing track, which incorporates 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 (GBFS) of LAMA, denoted as $alt([h^{\text{FF}}], \mathit{pref}\left(h^{\text{FF}}\right), [h^{\text{LC}}], \mathit{pref}\left(h^{\text{LC}}\right))$, where h^{LC} denotes the landmark-count heuristic and $\mathit{pref}(X)$ denotes the preferred operator queue with sorting strategy X. Type-LAMA (Xie et al. 2014) is an enhancement of LAMA which had the best coverage among single-algorithm (i.e., non-portfolio) planners in IPC2014. It adds Type-based buckets $\langle g, h\rangle$ to the list of alternating queues in LAMA: $alt\left([h^{\text{FF}}], \ldots, \mathit{pref}\left(h^{\text{LC}}\right), \left\langle g, h^{\text{FF}}\right\rangle\right)$.

We combine LAMA with intra/inter-plateau bucket/IP-diversification. The resulting configuration, LAMA-Btdb, incorporates all enhancements proposed in this paper: $alt\left([h^{\text{FF}}, \langle d\rangle], \mathit{pref}\left(h^{\text{FF}}\right), [h^{\text{LC}}, r_{\text{BIP}}], \mathit{pref}\left(h^{\text{LC}}\right), alt([r_{\text{BIP}}], \left\langle g, h^{\text{FF}}\right\rangle)\right)$. For intra-plateau diversification, we used depth for h^{FF} and IP for h^{LC} because we found different heuristics produce different plateau structures. For inter-plateau diversification, Btdb spends 1/5 of the entire search time (same frequency as those with which Type-LAMA selects from $\left\langle g, h^{\text{FF}}\right\rangle$) alternating bucket- and IP-diversification (i.e., $alt([r_{\text{BIP}}], \left\langle g, h^{\text{FF}}\right\rangle)$, spending 1/10 of the time for each queue). Adopting more sophisticated approaches for determining exploration frequency (Schulte and Keller 2014; Nakhost and Müller 2009) is a direction for future work.

Table 1 **(3)** shows that Btdb outperforms LAMA and Type-LAMA in the 5-minutes experiments. In IPC2011, Btdb (240.9) outperforms LAMA (235) and is comparable to Type-LAMA (241.2), while in IPC2014, Btdb (158.8) outperforms both LAMA (151.2) and Type-LAMA (145.9). Type-based exploration sometimes significantly degrades LAMA performance when the heuristic is correct but ignored in favor of the type-based queue. In contrast, Btdb, which integrates several diversification methods, ameliorates this drawback, resulting in the highest coverage.

6 Conclusion

In this paper, we first introduced the notion of *Intra-* and *Inter*-plateau diversification in satisficing heuristic search. We reformulated two previous diversification approaches, depth-diversification and Type-GBFS, as instances of a bucket-based diversification strategy applied to inter-plateau and intra-plateau diversification, respectively. We then showed empirically that these two modes of diversification have orthogonal, complementary effects.

Next, we showed that bucket-based diversification is not sufficient for bias removal in graphs where nodes have largely varying number of neighbors, and proposed IP-diversification, a new breadth-aware diversification strategy which addresses this issue. We evaluated the baseline performance of these approaches using GBFS. We showed that a planner which combines all of the new methods yields state-of-the-art performance on IPC benchmark instances.

This study has shown that our diversification methods eliminates some search biases, yet there could be more. A general, principled framework for quantifying search biases, perhaps incorporating notions such as partial order and symmetry (Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2013; Hall et al. 2013; Wehrle et al. 2013), is a direction for future work.

References

Asai, M., and Fukunaga, A. 2016. Tiebreaking Strategies for Classical Planning Using A^* Search. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Barabási, A.-L. 1996. Invasion percolation and global optimization. *Physical Review Letters* 76(20):3750.

Chu, Y.-J., and Liu, T.-H. 1965. On shortest arborescence of a directed graph. *Scientia Sinica* 14(10):1396.

Domshlak, C.; Katz, M.; and 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.

Felner, A.; Kraus, S.; and Korf, R. E. 2003. KBFS: K-best-first search. *Annals of Mathematics and Artificial Intelligence* 39(1-2):19–39.

Gabow, H. N.; Galil, Z.; Spencer, T.; and Tarjan, R. E. 1986. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica* 6(2):109–122.

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

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.

Imai, T., and 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*.

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

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

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

Prim, R. C. 1957. Shortest connection networks and some generalizations. *Bell system technical journal* 36(6):1389–1401.

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.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks Revisited. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Röger, G., and 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., and Keller, T. 2014. Balancing Exploration and Exploitation in Classical Planning. In *Proceedings of Annual Symposium on Combinatorial Search*.

Tarjan, R. E. 1977. Finding optimum branchings. *Networks* 7(1):25–35.

Valenzano, R. A., and Xie, F. 2016. On the Completeness of BestFirst Search Variants that Use Random Exploration. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Valenzano, R. A.; Schaeffer, J.; Sturtevant, N.; and 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)*.

Wehrle, M.; Helmert, M.; Alkhazraji, Y.; and 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., and Willemsen, J. F. 1983. Invasion percolation: a new form of percolation theory. *Journal of Physics A: Mathematical and General* 16(14):3365.

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

Improving Greedy Best-First Search by Removing Unintended Search Bias

Submision 1537

Evaluation under Blind Search

We also compared the blind search performance with and without our 2 diversification schemes in order to assess their pure effect not affected by any particular heuristics. We compared the performance of (h) the standard GBFS $[h]$ and its enhancements: (hd) bucket diversification $[h, \langle d \rangle]$, (hb) IP diversification $[h, r_{\text{BIP}}]$. Due to the characteristics of Blind heuristics, the only meaningful diversification is intra-plateau diversification: It always returns a constant value 1.

The results show that both (hd) and (hb) improves the performance of blind search while not strictly dominating each other: (hb) shows better performance than (hd) on Tidybot domain. The improvements are domain-dependent and we did not observe difference in domains not listed in the table.

Finally, we also compared the simple random selection (*ro*) with a selection based on fixed random values (hb). The results indicate that the baseline performance of IP-diversification and *ro* default tiebreaking are different 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 S1: Blind search results of 3 minutes runs (average of 4 runs). (hd) and (hb) are not dominating each other and (hb) and (*ro*) are different.

	h^{CG}									h^{FF}								
				fifo				*lifo*	*ro*				*fifo*				*lifo*	*ro*
domain	h	hd	ht(g,h)	ht(g)	ht(h)	hb	hB	h	h	h	hd	ht(g,h)	ht(g)	hb	hB	ht(h)	h	h
ipc2011	151.0	157.3	160.8	160.8	156.0	158.0	**175.0**	150.5	155.3	158.3	169.0	171.5	173.0	170.5	**198.3**	164.5	159.8	170.0
barman	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.0	8.3	10.5	9.0	8.3	**16.8**	8.0	0.8	8.5
elevators	9.0	8.0	8.8	8.8	9.0	9.8	**13.3**	8.0	8.3	18.5	14.0	16.0	17.0	18.0	**19.0**	15.3	10.0	16.8
floortile	0.0	0.0	**2.3**	2.0	1.8	0.0	0.5	0.0	0.0	5.8	6.3	7.3	**7.8**	4.3	5.3	7.3	5.0	4.8
nomystery	7.0	6.0	**15.8**	8.0	8.0	6.3	5.5	7.0	7.0	9.0	7.0	**16.5**	9.0	6.8	7.5	9.3	6.0	7.5
openstacks	10.0	**14.8**	10.0	10.8	10.0	12.0	13.5	13.8	12.0	11.0	18.3	11.0	13.0	19.5	17.3	13.0	19.0	**19.8**
parcprinter	**20.0**	**20.0**	19.5	16.8	18.3	19.8	13.0	**20.0**	19.5	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
parking	18.0	18.0	11.5	15.5	14.8	**19.8**	**19.8**	15.0	**19.8**	10.5	**20.0**	9.8	10.3	17.0	13.3	9.3	**20.0**	16.5
pegsol	20.0	20.0	20.0	20.0	20.0	20.0	19.8	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
scanalyzer	20.0	20.0	20.0	20.0	19.8	20.0	20.0	20.0	20.0	15.5	15.3	17.8	**19.0**	16.8	18.8	18.3	17.0	16.3
sokoban	16.0	16.0	16.8	16.3	**17.0**	15.8	16.0	15.0	16.0	**19.0**	**19.0**	17.3	17.3	18.5	18.3	17.8	**19.0**	18.5
tidybot	16.0	18.0	**18.3**	15.0	**18.3**	18.0	16.8	14.0	17.3	16.0	16.0	16.0	16.0	14.8	**16.3**	14.8	16.0	15.3
transport	10.0	11.5	9.0	10.3	8.5	11.0	12.3	**13.0**	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
visitall	3.0	3.0	6.3	8.8	5.5	3.8	**10.0**	3.8	4.0	3.0	3.0	6.0	7.8	5.3	**11.3**	6.8	5.0	4.5
woodworking	2.0	2.0	2.8	8.8	5.3	2.0	**14.8**	1.0	1.5	2.0	2.0	3.5	7.0	1.5	**14.8**	5.0	2.0	1.8
ipc2014	76.5	85.0	85.5	72.5	73.8	77.0	**88.3**	69.8	80.5	71.5	95.3	78.8	70.3	91.0	98.3	67.5	**98.5**	90.0
barman	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.3	0.0	**7.8**	0.0	0.0	0.0
cavediving	7.0	7.0	7.0	7.0	**7.3**	**7.3**	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	**7.3**	7.0	7.0
childsnack	1.0	**6.0**	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	**4.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0
citycar	0.0	0.0	**8.3**	3.0	0.0	0.3	1.3	0.0	3.8	0.0	0.0	**7.0**	3.8	0.0	2.8	1.8	0.0	0.0
floortile	0.0	0.0	**2.0**	1.8	1.8	0.0	0.3	0.0	0.0	2.0	2.0	2.0	2.0	2.0	**2.3**	2.0	2.0	2.0
ged	0.0	0.0	**9.5**	0.3	6.8	0.0	4.3	0.0	0.0	19.0	19.0	13.8	11.5	18.8	12.5	15.0	19.0	**20.0**
hiking	18.0	16.8	**19.5**	18.0	18.3	15.8	18.3	16.0	15.8	**20.0**	**20.0**	19.8	19.3	18.0	**20.0**	19.5	17.0	17.8
maintenance	16.0	16.0	**16.3**	15.3	15.8	14.0	15.8	16.0	14.8	11.0	8.0	10.8	10.5	6.8	10.8	8.3	**12.0**	8.0
openstacks	0.0	**3.5**	0.0	0.0	0.0	0.0	2.3	2.5	0.0	0.0	12.3	0.0	0.8	**14.8**	11.3	0.3	14.3	14.5
parking	6.8	9.8	0.8	3.3	2.0	**11.3**	8.3	2.0	10.0	3.3	7.3	1.3	1.3	5.8	1.5	2.0	**12.3**	3.5
tetris	17.8	18.0	14.0	14.3	12.5	**20.0**	18.0	14.0	19.5	1.3	6.8	3.3	3.5	**9.3**	8.3	1.3	4.0	8.0
thoughtful	5.0	5.0	5.0	5.0	5.0	4.5	5.0	5.0	5.5	8.0	9.0	**13.0**	10.5	8.8	11.0	10.3	11.0	9.3
transport	5.0	3.0	3.3	4.8	4.5	4.0	**6.0**	5.3	4.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
visitall	0.0	0.0	0.0	0.0	0.0	0.0	**2.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**3.3**	0.0	0.0	0.0

Table S2: Eager GBFS results (extended). Average of 4 runs, 5 minutes time limit with 4GB memory limit.

	h^{CG}									h^{FF}								
				fifo				*lifo*	*ro*				*fifo*				*lifo*	*ro*
domain	h	hd	ht(g,h)	ht(g)	ht(h)	hb	hB	h	h	h	hd	ht(g,h)	ht(g)	ht(h)	hb	hB	h	h
ipc2011	151	156	153	151	143	150	**167**	149	151	156	168	163	164	157	162	**187**	159	158
barman	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.0	8.0	10.0	9.0	6.0	7.0	**15.0**	0.0	8.0
elevators	9.0	8.0	8.0	8.0	8.0	9.0	**13.0**	8.0	7.0	**18.0**	14.0	15.0	16.0	15.0	17.0	**18.0**	10.0	14.0
floortile	0.0	0.0	**2.0**	**2.0**	1.0	0.0	0.0	0.0	0.0	5.0	6.0	6.0	6.0	**7.0**	4.0	4.0	5.0	4.0
nomystery	7.0	6.0	**15.0**	7.0	7.0	6.0	5.0	7.0	7.0	9.0	7.0	**16.0**	9.0	9.0	6.0	7.0	6.0	6.0
openstacks	10.0	**14.0**	10.0	10.0	10.0	12.0	13.0	13.0	12.0	11.0	18.0	11.0	13.0	13.0	**19.0**	17.0	**19.0**	**19.0**
parcprinter	**20.0**	**20.0**	19.0	15.0	15.0	19.0	12.0	**20.0**	19.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
parking	18.0	18.0	10.0	14.0	13.0	**19.0**	**19.0**	15.0	**19.0**	10.0	**20.0**	8.0	8.0	8.0	16.0	12.0	**20.0**	14.0
pegsol	20.0	20.0	20.0	20.0	20.0	20.0	19.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
scanalyzer	20.0	20.0	20.0	20.0	19.0	20.0	20.0	20.0	20.0	15.0	15.0	17.0	**18.0**	**18.0**	16.0	**18.0**	17.0	16.0
sokoban	16.0	16.0	16.0	16.0	16.0	15.0	16.0	15.0	16.0	**19.0**	**19.0**	17.0	17.0	17.0	18.0	17.0	19.0	18.0
tidybot	16.0	**18.0**	17.0	13.0	17.0	17.0	16.0	14.0	17.0	**16.0**	**16.0**	15.0	15.0	14.0	14.0	14.0	**16.0**	14.0
transport	10.0	11.0	8.0	10.0	8.0	9.0	11.0	**13.0**	9.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
visitall	3.0	3.0	6.0	8.0	5.0	3.0	**10.0**	3.0	4.0	3.0	3.0	5.0	7.0	6.0	4.0	**11.0**	5.0	4.0
woodworking	2.0	2.0	2.0	8.0	4.0	1.0	**13.0**	1.0	1.0	2.0	2.0	3.0	6.0	4.0	1.0	**14.0**	2.0	1.0
ipc2014	75	**83**	78	66	62	72	80	69	71	71	93	70	63	59	82	88	**98**	78
barman	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**7.0**	0.0	0.0
cavediving	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0
childsnack	1.0	**6.0**	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	**4.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0
citycar	0.0	0.0	6.0	2.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	**6.0**	3.0	1.0	0.0	2.0	0.0	0.0
floortile	0.0	0.0	**2.0**	1.0	1.0	0.0	0.0	0.0	0.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
ged	0.0	0.0	**8.0**	0.0	5.0	0.0	3.0	0.0	0.0	19.0	19.0	13.0	11.0	14.0	18.0	12.0	19.0	**20.0**
hiking	18.0	16.0	**19.0**	17.0	17.0	15.0	17.0	16.0	14.0	**20.0**	**20.0**	19.0	18.0	19.0	17.0	**20.0**	17.0	16.0
maintenance	**16.0**	**16.0**	15.0	14.0	15.0	12.0	15.0	**16.0**	12.0	**11.0**	8.0	10.0	9.0	6.0	5.0	10.0	12.0	5.0
openstacks	0.0	**3.0**	0.0	0.0	0.0	0.0	2.0	2.0	0.0	0.0	12.0	0.0	0.0	0.0	**14.0**	9.0	**14.0**	13.0
parking	6.0	9.0	0.0	3.0	0.0	**10.0**	6.0	2.0	9.0	3.0	6.0	0.0	0.0	1.0	4.0	0.0	**12.0**	1.0
tetris	17.0	18.0	13.0	13.0	10.0	**20.0**	17.0	14.0	19.0	1.0	6.0	2.0	3.0	1.0	**8.0**	7.0	4.0	6.0
thoughtful	5.0	5.0	5.0	5.0	5.0	4.0	5.0	5.0	5.0	8.0	9.0	**11.0**	10.0	8.0	7.0	10.0	**11.0**	8.0
transport	5.0	3.0	3.0	4.0	2.0	4.0	**6.0**	5.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
visitall	0.0	0.0	0.0	0.0	0.0	0.0	**2.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**2.0**	0.0	0.0

Table S3: Eager GBFS results (extended). **Minimum** of 4 runs, 5 minutes time limit with 4GB memory limit.

	h^{CG}									h^{FF}								
				fifo				*lifo*	*ro*				*fifo*				*lifo*	*ro*
domain	h	hd	ht(g,h)	ht(g)	ht(h)	hb	hB	h	h	h	hd	ht(g,h)	ht(g)	ht(h)	hb	hB	h	h
ipc2011	151	158	168	172	165	164	**182**	151	159	160	172	183	182	174	178	**210**	160	180
barman	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.0	9.0	12.0	9.0	9.0	9.0	**18.0**	1.0	9.0
elevators	9.0	8.0	9.0	9.0	10.0	10.0	**14.0**	8.0	9.0	19.0	14.0	17.0	18.0	16.0	19.0	**20.0**	10.0	18.0
floortile	0.0	0.0	**3.0**	2.0	2.0	0.0	1.0	0.0	0.0	6.0	7.0	8.0	**9.0**	8.0	5.0	7.0	5.0	6.0
nomystery	7.0	6.0	**16.0**	9.0	9.0	7.0	6.0	7.0	7.0	9.0	7.0	**17.0**	9.0	10.0	7.0	8.0	6.0	9.0
openstacks	10.0	**15.0**	10.0	11.0	10.0	12.0	14.0	14.0	12.0	11.0	19.0	11.0	13.0	13.0	**20.0**	18.0	19.0	**20.0**
parcprinter	**20.0**	**20.0**	**20.0**	18.0	**20.0**	**20.0**	14.0	**20.0**	**20.0**	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
parking	18.0	18.0	13.0	18.0	16.0	**20.0**	**20.0**	15.0	**20.0**	11.0	**20.0**	12.0	13.0	12.0	18.0	15.0	**20.0**	19.0
pegsol	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
scanalyzer	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	16.0	16.0	19.0	**20.0**	19.0	17.0	**20.0**	17.0	17.0
sokoban	16.0	16.0	17.0	17.0	**18.0**	16.0	16.0	15.0	16.0	19.0	19.0	18.0	18.0	18.0	19.0	19.0	19.0	19.0
tidybot	16.0	18.0	**19.0**	18.0	**19.0**	**19.0**	18.0	14.0	18.0	16.0	16.0	17.0	17.0	16.0	15.0	**18.0**	16.0	16.0
transport	10.0	12.0	10.0	11.0	9.0	**13.0**	**13.0**	**13.0**	11.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
visitall	3.0	3.0	7.0	9.0	6.0	4.0	**10.0**	4.0	4.0	3.0	3.0	7.0	8.0	7.0	7.0	**12.0**	5.0	5.0
woodworking	2.0	2.0	4.0	10.0	6.0	3.0	**16.0**	1.0	2.0	2.0	2.0	5.0	8.0	6.0	2.0	**15.0**	2.0	2.0
ipc2014	77	86	92	81	83	82	**97**	71	89	73	97	90	79	77	100	**108**	100	99
barman	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	1.0	0.0	0.0	**8.0**	0.0	0.0
cavediving	7.0	7.0	7.0	7.0	8.0	8.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0	7.0	7.0	7.0	7.0
childsnack	1.0	6.0	0.0	0.0	0.0	0.0	0.0	**2.0**	0.0	0.0	**4.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0
citycar	0.0	0.0	**9.0**	5.0	0.0	1.0	2.0	0.0	5.0	0.0	0.0	**9.0**	5.0	3.0	0.0	4.0	0.0	0.0
floortile	0.0	0.0	2.0	2.0	2.0	0.0	1.0	0.0	0.0	2.0	2.0	2.0	2.0	2.0	2.0	3.0	2.0	2.0
ged	0.0	0.0	**11.0**	1.0	8.0	0.0	5.0	0.0	0.0	19.0	19.0	15.0	13.0	16.0	19.0	13.0	19.0	**20.0**
hiking	18.0	17.0	**20.0**	**20.0**	**20.0**	16.0	19.0	16.0	17.0	**20.0**	**20.0**	**20.0**	**20.0**	**20.0**	19.0	**20.0**	17.0	19.0
maintenance	16.0	16.0	17.0	16.0	16.0	16.0	17.0	16.0	17.0	11.0	8.0	**12.0**	11.0	10.0	9.0	11.0	**12.0**	10.0
openstacks	0.0	**4.0**	0.0	0.0	0.0	0.0	3.0	3.0	0.0	0.0	13.0	0.0	1.0	1.0	**16.0**	13.0	15.0	**16.0**
parking	7.0	10.0	1.0	4.0	4.0	**12.0**	11.0	2.0	**12.0**	4.0	8.0	3.0	4.0	3.0	8.0	3.0	**13.0**	5.0
tetris	18.0	18.0	16.0	16.0	14.0	**20.0**	19.0	14.0	**20.0**	2.0	7.0	5.0	4.0	2.0	**10.0**	9.0	4.0	9.0
thoughtful	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	6.0	8.0	9.0	**15.0**	11.0	12.0	10.0	13.0	11.0	11.0
transport	5.0	3.0	4.0	5.0	**6.0**	4.0	**6.0**	**6.0**	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
visitall	0.0	0.0	0.0	0.0	0.0	0.0	**2.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**4.0**	0.0	0.0

Table S4: Eager GBFS results (extended). **Maximum** of 4 runs, 5 minutes time limit with 4GB memory limit.

	h^{CG}								h^{FF}							
	fifo						*lifo*	*ro*	*fifo*						*lifo*	*ro*
domain	h	hd	ht(g,h)	ht(h)	hb	hB	h	h	h	hd	ht(g,h)	ht(h)	hb	hB	h	h
ipc2011	141.0	152.8	149.8	151.8	153.8	**179.8**	121.3	150.8	172.0	177.0	190.0	179.5	163.3	**202.3**	123.0	163.3
barman	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.0	8.0	9.8	8.0	7.5	**17.8**	2.0	8.5
elevators	7.0	8.0	8.5	8.0	9.3	**14.0**	9.0	9.0	17.5	12.3	14.8	16.0	16.5	**18.0**	8.0	**18.0**
floortile	0.0	0.0	1.0	**1.5**	0.0	0.3	0.0	0.0	4.0	4.0	**7.0**	**7.0**	4.0	4.5	4.0	4.8
nomystery	5.0	5.0	**14.0**	7.0	5.0	5.3	10.0	5.5	7.0	7.0	**16.5**	8.3	5.0	6.5	5.0	5.5
openstacks	15.0	**20.0**	14.8	16.0	16.8	17.0	0.0	18.3	**20.0**	**20.0**	**20.0**	**20.0**	18.8	**20.0**	0.0	**20.0**
parcprinter	**20.0**	**20.0**	18.8	19.0	**20.0**	19.0	11.0	10.0	**20.0**	18.0	19.8	19.5	14.5	18.5	11.0	8.8
parking	11.5	11.8	4.5	9.8	16.5	16.5	3.8	**19.0**	7.5	18.8	8.8	10.3	11.8	12.3	17.0	**19.3**
pegsol	20.0	20.0	20.0	19.8	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
scanalyzer	20.0	20.0	19.0	19.3	20.0	20.0	20.0	20.0	16.0	16.0	**18.8**	18.5	16.8	**18.8**	16.0	16.8
sokoban	15.0	14.3	16.3	16.5	13.8	14.8	**17.0**	15.8	**19.0**	**19.0**	17.5	17.5	18.5	18.5	**19.0**	18.5
tidybot	15.0	**19.0**	18.0	16.3	18.3	17.3	14.0	17.5	16.0	16.0	15.8	16.0	14.3	**17.3**	14.0	15.3
transport	8.5	10.8	7.3	9.0	10.3	11.8	**12.0**	10.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
visitall	3.0	3.0	6.0	5.0	3.0	**10.0**	3.5	4.0	3.0	4.0	6.5	6.0	5.3	**11.5**	5.0	4.8
woodworking	1.0	1.0	1.8	4.8	1.0	**14.0**	1.0	1.3	14.0	14.0	15.0	12.5	10.5	**18.8**	2.0	3.3
ipc2014	58.8	73.3	75.3	64.5	71.8	**83.8**	44.0	74.5	84.0	90.5	**98.8**	85.3	85.8	92.8	63.8	95.8
barman	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.0	**5.8**	0.0	0.0
cavediving	7.0	7.0	7.0	**7.3**	7.0	7.0	7.0	7.0	7.0	7.0	**7.3**	**7.3**	7.0	7.0	7.0	7.0
childsnack	0.0	0.0	0.0	0.0	0.0	0.0	**1.0**	0.0	0.0	**4.0**	0.3	0.0	0.0	0.0	0.0	0.0
citycar	4.0	0.0	**4.5**	0.3	0.0	2.0	0.0	1.0	0.0	0.0	**10.8**	2.0	0.0	2.5	0.0	0.0
floortile	0.0	0.0	1.3	**2.0**	0.0	0.0	0.0	0.0	2.0	2.0	**2.3**	2.0	2.0	**2.3**	2.0	2.0
ged	0.0	0.0	**11.0**	7.5	0.0	7.8	0.0	0.0	19.0	19.0	15.8	15.8	19.8	7.5	**20.0**	19.5
hiking	15.0	16.0	18.5	**18.8**	15.5	18.3	17.0	15.5	**20.0**	**20.0**	19.5	**20.0**	18.8	**20.0**	13.0	17.3
maintenance	1.0	0.0	**6.8**	0.8	0.0	0.8	0.0	0.5	5.0	0.0	**7.8**	6.0	0.5	4.3	4.0	0.5
openstacks	5.8	**20.0**	4.8	7.8	16.0	11.3	0.0	13.8	18.0	**20.0**	17.5	17.5	18.5	19.3	0.0	**20.0**
parking	2.0	3.3	0.0	0.3	5.3	5.8	0.0	**7.0**	2.0	5.3	0.3	3.0	2.5	2.3	7.8	**13.0**
tetris	16.0	18.0	14.0	12.0	19.8	17.8	9.0	**20.0**	2.0	5.3	6.3	1.3	**9.5**	7.0	3.0	8.8
thoughtful	5.0	5.0	5.0	5.0	4.5	5.3	**6.0**	5.5	9.0	8.0	11.0	10.5	7.3	**11.3**	7.0	7.8
transport	3.0	4.0	2.5	3.0	3.8	**6.0**	4.0	4.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
visitall	0.0	0.0	0.0	0.0	0.0	**2.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**3.8**	0.0	0.0

Table S5: Lazy GBFS results. Average of 4 runs, 5 minutes time limit with 4GB memory limit.

	h^{CG}								h^{FF}							
	fifo						*lifo*	*ro*	*fifo*						*lifo*	*ro*
domain	h	hd	ht(g,h)	ht(h)	hb	hB	h	h	h	hd	ht(g,h)	ht(h)	hb	hB	h	h
ipc2011	140	151	141	142	147	**172**	120	141	171	176	179	171	153	**190**	123	154
barman	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.0	8.0	9.0	7.0	7.0	**17.0**	2.0	8.0
elevators	7.0	8.0	8.0	8.0	8.0	**14.0**	9.0	8.0	**17.0**	12.0	14.0	15.0	16.0	**17.0**	8.0	**17.0**
floortile	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	4.0	**7.0**	6.0	3.0	3.0	4.0	4.0
nomystery	5.0	5.0	**14.0**	7.0	5.0	5.0	10.0	5.0	7.0	7.0	**16.0**	8.0	5.0	5.0	5.0	5.0
openstacks	15.0	**20.0**	14.0	16.0	15.0	16.0	0.0	18.0	**20.0**	**20.0**	**20.0**	**20.0**	17.0	**20.0**	0.0	**20.0**
parcprinter	**20.0**	**20.0**	18.0	17.0	**20.0**	18.0	11.0	9.0	**20.0**	18.0	19.0	19.0	14.0	16.0	11.0	7.0
parking	11.0	11.0	3.0	9.0	16.0	14.0	3.0	**18.0**	7.0	**18.0**	7.0	9.0	11.0	11.0	17.0	**18.0**
pegsol	20.0	20.0	20.0	19.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
scanalyzer	20.0	20.0	18.0	19.0	20.0	20.0	20.0	20.0	16.0	16.0	17.0	**18.0**	16.0	**18.0**	16.0	16.0
sokoban	15.0	14.0	16.0	16.0	13.0	14.0	**17.0**	14.0	**19.0**	**19.0**	17.0	17.0	18.0	18.0	**19.0**	18.0
tidybot	15.0	**19.0**	17.0	15.0	18.0	17.0	14.0	16.0	**16.0**	**16.0**	15.0	15.0	13.0	**16.0**	14.0	15.0
transport	8.0	10.0	6.0	8.0	8.0	11.0	**12.0**	8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
visitall	3.0	3.0	6.0	5.0	3.0	**10.0**	3.0	4.0	3.0	4.0	5.0	5.0	4.0	**11.0**	5.0	4.0
woodworking	1.0	1.0	1.0	3.0	1.0	**13.0**	1.0	1.0	14.0	14.0	13.0	12.0	9.0	**18.0**	2.0	2.0
ipc2014	56	73	68	59	67	**77**	44	70	83	**89**	**89**	77	80	84	63	88
barman	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**5.0**	0.0	0.0
cavediving	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0
childsnack	0.0	0.0	0.0	0.0	0.0	0.0	**1.0**	0.0	0.0	**4.0**	0.0	0.0	0.0	0.0	0.0	0.0
citycar	**4.0**	0.0	**4.0**	0.0	0.0	1.0	0.0	1.0	0.0	0.0	**10.0**	1.0	0.0	2.0	0.0	0.0
floortile	0.0	0.0	1.0	**2.0**	0.0	0.0	0.0	0.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
ged	0.0	0.0	**10.0**	7.0	0.0	7.0	0.0	0.0	19.0	19.0	14.0	14.0	19.0	6.0	**20.0**	19.0
hiking	15.0	16.0	17.0	**18.0**	15.0	16.0	17.0	14.0	**20.0**	**20.0**	19.0	**20.0**	18.0	**20.0**	13.0	16.0
maintenance	1.0	0.0	**6.0**	0.0	0.0	0.0	0.0	0.0	5.0	0.0	**7.0**	4.0	0.0	3.0	4.0	0.0
openstacks	5.0	20.0	4.0	7.0	**15.0**	11.0	0.0	13.0	18.0	**20.0**	17.0	16.0	18.0	19.0	0.0	**20.0**
parking	0.0	3.0	0.0	0.0	4.0	5.0	0.0	**6.0**	2.0	5.0	0.0	2.0	2.0	1.0	7.0	**11.0**
tetris	16.0	18.0	13.0	11.0	19.0	17.0	9.0	**20.0**	1.0	4.0	3.0	1.0	**9.0**	6.0	3.0	8.0
thoughtful	5.0	5.0	5.0	5.0	4.0	5.0	**6.0**	5.0	9.0	8.0	**10.0**	**10.0**	5.0	**10.0**	7.0	5.0
transport	3.0	4.0	1.0	2.0	3.0	**6.0**	4.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
visitall	0.0	0.0	0.0	0.0	0.0	**2.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**3.0**	0.0	0.0

Table S6: Lazy GBFS results (extended). **Minimum** of 4 runs, 5 minutes time limit with 4GB memory limit.

	h^{CG}								h^{FF}							
	fifo						*lifo*	*ro*	*fifo*						*lifo*	*ro*
domain	h	hd	ht(g,h)	ht(h)	hb	hB	h	h	h	hd	ht(g,h)	ht(h)	hb	hB	h	h
ipc2011	142	154	158	159	162	**188**	122	160	173	178	199	189	175	**213**	123	173
barman	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.0	8.0	10.0	9.0	8.0	**19.0**	2.0	9.0
elevators	7.0	8.0	9.0	8.0	11.0	**14.0**	9.0	10.0	18.0	13.0	16.0	17.0	18.0	**19.0**	8.0	**19.0**
floortile	0.0	0.0	**2.0**	**2.0**	0.0	1.0	0.0	0.0	4.0	4.0	7.0	**8.0**	5.0	6.0	4.0	5.0
nomystery	5.0	5.0	**14.0**	7.0	5.0	6.0	10.0	6.0	7.0	7.0	**17.0**	9.0	5.0	7.0	5.0	6.0
openstacks	15.0	**20.0**	15.0	16.0	18.0	18.0	0.0	19.0	**20.0**	**20.0**	**20.0**	**20.0**	**20.0**	**20.0**	0.0	**20.0**
parcprinter	**20.0**	**20.0**	**20.0**	**20.0**	**20.0**	**20.0**	11.0	11.0	**20.0**	18.0	**20.0**	20.0	15.0	**20.0**	11.0	11.0
parking	12.0	12.0	5.0	11.0	17.0	18.0	4.0	**20.0**	8.0	19.0	11.0	12.0	14.0	13.0	17.0	**20.0**
pegsol	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
scanalyzer	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	16.0	16.0	**20.0**	19.0	17.0	**20.0**	16.0	17.0
sokoban	15.0	15.0	**17.0**	**17.0**	15.0	16.0	**17.0**	**17.0**	19.0	19.0	18.0	18.0	19.0	19.0	19.0	19.0
tidybot	15.0	19.0	**20.0**	17.0	19.0	18.0	14.0	18.0	16.0	16.0	16.0	17.0	15.0	**19.0**	14.0	16.0
transport	9.0	11.0	8.0	10.0	**13.0**	12.0	12.0	**13.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
visitall	3.0	3.0	6.0	5.0	3.0	**10.0**	4.0	4.0	3.0	4.0	7.0	7.0	6.0	**12.0**	5.0	6.0
woodworking	1.0	1.0	2.0	6.0	1.0	**15.0**	1.0	2.0	14.0	14.0	17.0	13.0	13.0	**19.0**	2.0	5.0
ipc2014	60	74	83	71	77	**91**	44	79	85	92	**111**	94	93	103	64	105
barman	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	**6.0**	0.0	0.0
cavediving	7.0	7.0	7.0	8.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0	8.0	7.0	7.0	7.0	7.0
childsnack	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	**4.0**	1.0	0.0	0.0	0.0	0.0	0.0
citycar	4.0	0.0	**5.0**	1.0	0.0	3.0	0.0	1.0	0.0	0.0	**13.0**	3.0	0.0	3.0	0.0	0.0
floortile	0.0	0.0	**2.0**	**2.0**	0.0	0.0	0.0	0.0	2.0	2.0	3.0	2.0	2.0	3.0	2.0	2.0
ged	0.0	0.0	**13.0**	9.0	0.0	9.0	0.0	0.0	19.0	19.0	17.0	17.0	**20.0**	9.0	**20.0**	**20.0**
hiking	15.0	16.0	**20.0**	19.0	16.0	20.0	17.0	16.0	**20.0**	**20.0**	**20.0**	**20.0**	**20.0**	**20.0**	13.0	19.0
maintenance	1.0	0.0	**8.0**	1.0	0.0	1.0	0.0	1.0	5.0	0.0	**9.0**	8.0	2.0	6.0	4.0	1.0
openstacks	6.0	**20.0**	5.0	8.0	17.0	12.0	0.0	15.0	18.0	**20.0**	18.0	19.0	19.0	**20.0**	0.0	**20.0**
parking	3.0	4.0	0.0	1.0	7.0	7.0	0.0	**8.0**	2.0	6.0	1.0	4.0	4.0	5.0	8.0	**15.0**
tetris	16.0	18.0	15.0	13.0	**20.0**	18.0	9.0	**20.0**	3.0	6.0	8.0	2.0	**10.0**	8.0	3.0	**10.0**
thoughtful	5.0	5.0	5.0	5.0	5.0	6.0	6.0	6.0	9.0	8.0	**12.0**	11.0	9.0	**12.0**	7.0	11.0
transport	3.0	4.0	3.0	4.0	5.0	**6.0**	4.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
visitall	0.0	0.0	0.0	0.0	0.0	**2.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**4.0**	0.0	0.0

Table S7: Lazy GBFS results (extended). **Maximum** of 4 runs, 5 minutes time limit with 4GB memory limit.

	h^{CG}						h^{FF}					
diversification	none	intra	inter	intra+inter	mix	intra+mix	none	intra	inter	intra+inter	mix	intra+mix
domain	**h**	**hd**	**ht(h)**	**hdt(h)**	**ht(g,h)**	**hdt(g,h)**	**h**	**hd**	**ht(h)**	**hdt(h)**	**ht(g,h)**	**hdt(g,h)**
ipc2011	151.00	157.25	156.00	160.25	160.75	174.50	158.25	169.00	164.50	175.25	171.50	186.00
barman	0	0	0	0	0	0	8	8.25	8	8.25	10.5	10
elevators	9	8	9	9	8.75	9.5	18.5	14	15.25	14.5	16	14.25
floortile	0	0	**1.75**	**1.75**	**2.25**	**2**	5.75	6.25	**7.25**	**7.5**	**7.25**	**7**
nomystery	7	6	**8**	**7.25**	**15.75**	**15.5**	9	7	9.25	8.75	**16.5**	**16.5**
openstacks	10	*14.75*	10	*13*	10	*12.75*	11	*18.25*	13	*17.25*	11	*16*
parcprinter	20	20	18.25	17.75	19.5	18	20	20	20	20	20	20
parking	18	18	14.75	14.25	11.5	13.5	10.5	*20*	9.25	*16.75*	9.75	*16.5*
pegsol	20	20	20	20	20	20	20	20	20	20	20	20
scanalyzer	20	20	19.75	20	20	20	15.5	15.25	**18.25**	**17.75**	**17.75**	**18.25**
sokoban	16	16	**17**	**17**	**16.75**	**16.75**	19	19	17.75	17.5	17.25	17.5
tidybot	16	18	18.25	17.75	18.25	19.75	16	16	14.75	15.25	16	16.5
transport	10	*11.5*	8.5	*11.5*	9	*12.25*	0	0	0	0	0	0.25
visitall	3	3	**5.5**	**5.25**	**6.25**	**6.25**	3	3	**6.75**	**5.5**	**6**	**6.25**
woodworking	2	2	**5.25**	**5.75**	**2.75**	**8.25**	2	2	**5**	**6.25**	**3.5**	**7**
ipc2014	76.50	85.00	73.75	81.75	85.50	91.00	71.50	95.25	67.50	84.00	78.75	93.25
barman	0	0	0	0	0	0	0	0	0	0	1	1
cavediving	7	7	7.25	7.5	7	7	7	7	7.25	7	7	7.25
childsnack	1	*6*	0	*1.5*	0	*2.25*	0	*4*	0	*0.25*	0	*0.5*
citycar	0	0	0	0	**8.25**	**4.25**	0	0	**1.75**	**2.25**	**7**	**7.5**
floortile	0	0	**1.75**	**1.5**	**2**	**2**	2	2	2	2	2	2.25
ged	0	0	**6.75**	**6.25**	**9.5**	**10.25**	19	19	15	15.5	13.75	13.25
hiking	18	16.75	**18.25**	**20**	**19.5**	**20**	20	20	19.5	20	19.75	20
maintenance	16	16	15.75	16	16.25	15.5	11	8	8.25	9	10.75	10.75
openstacks	0	*3.5*	0	*0.75*	0	*0.5*	0	*12.25*	0.25	*8*	0	*6.5*
parking	6.75	*9.75*	2	*3.5*	0.75	*3.75*	3.25	*7.25*	2	*5.5*	1.25	*4.75*
tetris	17.75	18	12.5	15.5	14	15.5	1.25	*6.75*	1.25	*4*	3.25	*6*
thoughtful	5	5	5	5	5	5	8	9	**10.25**	**10.5**	**13**	**13.5**
transport	5	3	4.5	4.25	3.25	5	0	0	0	0	0	0
visitall	0	0	0	0	0	0	0	0	0	0	0	0

Table S8: Comparison of (**hdt(h)**) $alt([h, \langle d \rangle], \langle h \rangle)$, (**hdt(g,h)**) $alt([h, \langle d \rangle], \langle g, h \rangle)$ and (**hd,ht(h),ht(g,h)**). Results are highlighted arbitrarily, but **bold** highlights the results improved by inter-plateau diversification, while *underlines* highlights the results improved by intra-plateau diversification.

	h^{CG}						h^{FF}					
diversification	intra hb	inter hB	intra hw	inter hW	intra hp	inter hP	intra hb	inter hB	intra hw	inter hW	intra hp	inter hP
IPC2011	158.00	175.00	155.50	154.75	151.00	168.25	170.50	198.25	164.50	171.25	170.50	175.50
barman	0	0	0	0	0	0	8.25	16.75	9	15.75	8	10.25
elevators	9.75	13.25	9	8	8	10	18	19	17	15.25	16	16.75
floortile	0	0.5	0	2.25	0	2	4.25	5.25	5	7	6	7
nomystery	6.25	5.5	7	7.5	6	8.25	6.75	7.5	8	7.75	6	9
openstacks	12	13.5	12	10	14.75	11	19.5	17.25	18.75	11	19	13
parcprinter	19.75	13	19	15.5	19	18	20	20	20	19.5	20	20
parking	19.75	19.75	19.75	17.5	16.75	16.25	17	13.25	10	11.25	18.75	10.25
pegsol	20	19.75	20	20	20	20	20	20	20	20	20	20
scanalyzer	20	20	20	20	20	20	16.75	18.75	15.75	17.75	15.75	18.75
sokoban	15.75	16	16	15.75	16	16.5	18.5	18.25	19	19	19	19
tidybot	18	16.75	17	17	15	17.25	14.75	16.25	15	16.5	16	16.25
transport	11	12.25	10.75	9	10.5	11.25	0	0	0	0	0	0
visitall	3.75	10	3	6	3	4.75	5.25	11.25	4	6.25	4	5.5
woodworking	2	14.75	2	6.25	2	13	1.5	14.75	3	4.25	2	9.75
IPC2014	77.00	88.25	86.50	75.00	83.50	74.25	91.00	98.25	89.00	73.50	92.50	67.50
barman	0	0	0	0	0	0	0	7.75	0	8.75	0	0.25
cavediving	7.25	7	7	7	7	7	7	7	6	7	7	7
childsnack	0	0	0	0	3	0.25	0	0	0	0	2	0
citycar	0.25	1.25	3	1.5	3.25	0.75	0	2.75	0	1.75	0	1.5
floortile	0	0.25	0	2	0	2	2	2.25	2	2	2	2
ged	0	4.25	0	3	0	3	18.75	12.5	20	7.75	19	12.5
hiking	15.75	18.25	18	19.25	18	18.25	18	20	20	19.25	20	20
maintenance	14	15.75	16	16	16	16.75	6.75	10.75	8	11	8	9.25
openstacks	0	2.25	1.75	0	3.25	0	14.75	11.25	12.25	0	12.5	1.5
parking	11.25	8.25	10.75	4	4.25	3.5	5.75	1.5	2.75	2.75	5.75	1.25
tetris	20	18	20	12.5	19.75	13.25	9.25	8.25	11	2.25	6.25	2
thoughtful	4.5	5	5	5	5	5.25	8.75	11	7	11	10	10.25
transport	4	6	5	4.75	4	4.25	0	0	0	0	0	0
visitall	0	2	0	0	0	0	0	3.25	0	0	0	0

Table S9: Supplement-only results (avg. 4 runs) of diversification using the number of successors, eager-GBFS. Let w be the number of successors of a node, and p be the number of siblings of a node (number of nodes generated by the same parents). (**hw**) is $[h, \langle w \rangle]$, (**hW**) is $alt(h, \langle w \rangle)$, (**hp**) is $[h, \langle p \rangle]$, (**hP**) is $alt(h, \langle p \rangle)$. These variants failed to compete with IP-diversification methods applied on intra- / inter-plateau diversification.

	h^{CG}						h^{FF}					
diversification	intra hb	inter hB	intra hw	inter hW	intra hp	inter hP	intra hb	inter hB	intra hw	inter hW	intra hp	inter hP
IPC2011	153.75	179.75	146.75	160.75	150.00	157.00	163.25	202.25	174.00	186.50	165.50	190.00
barman	0	0	0	0	0	0	7.5	17.75	8	9.25	8	17
elevators	9.25	14	7.75	8.75	8	8.25	16.5	18	15	15	16	16
floortile	0	0.25	0	1.5	0	2	4	4.5	4.25	6.75	4	6.5
nomystery	5	5.25	5	7.25	5	7.25	5	6.5	5	8.25	7	9.25
openstacks	16.75	17	20	15	16	16	18.75	20	20	19	20	20
parcprinter	20	19	20	19	20	16.5	14.5	18.5	18	20	18	18.5
parking	16.5	16.5	10	11.75	18.5	16.25	11.75	12.25	15.75	10.5	5.5	9.75
pegsol	20	20	20	20	20	20	20	20	20	20	20	20
scanalyzer	20	20	20	20	20	20	16.75	18.75	16	18.75	16	19
sokoban	13.75	14.75	15	16	15	16	18.5	18.5	19	18.75	19	19
tidybot	18.25	17.25	13	16.75	14	16.5	14.25	17.25	16	16.5	16	16.25
transport	10.25	11.75	12	9.75	9.5	10.25	0	0	0	0	0	0
visitall	3	10	3	4.75	3	5.5	5.25	11.5	4	5.25	4	6.75
woodworking	1	14	1	10.25	1	2.5	10.5	18.75	13	18.5	12	12
IPC2014	71.75	83.75	74.00	64.50	71.50	73.50	85.75	92.75	84.00	71.00	84.75	86.25
barman	0	0	0	0	0	0	0	5.75	0	1.25	0	11
cavediving	7	7	6	7	7	7.25	7	7	7	7	6	7.25
childsnack	0	0	0	0	0	0	0	0	0	0	0	0
citycar	0	2	4	2	3	2.5	0	2.5	0	1	0	1.25
floortile	0	0	0	2	0	2	2	2.25	2	2	2	2
ged	0	7.75	0	4.5	0	7.25	19.75	7.5	20	7.5	20	6.5
hiking	15.5	18.25	17	18.25	16	19.5	18.75	20	20	20	20	20
maintenance	0	0.75	0	0.5	0	0.75	0.5	4.25	0	3.75	0	4.75
openstacks	16	11.25	20	6.25	9.25	9	18.5	19.25	20	15.25	20	18
parking	5.25	5.75	0	1.25	7.25	4.25	2.5	2.25	5	1.5	1.75	2.75
tetris	19.75	17.75	19	12.75	20	11.5	9.5	7	4	1.5	8	2.5
thoughtful	4.5	5.25	4	5	5	5	7.25	11.25	6	10.25	7	10.25
transport	3.75	6	4	5	4	4.5	0	0	0	0	0	0
visitall	0	2	0	0	0	0	0	3.75	0	0	0	0

Table S10: Supplement-only results (avg. 4 runs) of diversification using the number of successors, lazy-GBFS. Let w be the number of successors of a node, and p be the number of siblings of a node (number of nodes generated by the same parents). (**hw**) is $[h, \langle w \rangle]$, (**hW**) is $alt(h, \langle w \rangle)$, (**hp**) is $[h, \langle p \rangle]$, (**hP**) is $alt(h, \langle p \rangle)$. These variants failed to compete with IP-diversification methods applied on intra- / inter-plateau diversification.