Biased Exploration for Satisficing Heuristic Search

Ryo Kuroiwa, J. Christopher Beck

Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada, ON M5S 3G8
ryo.kuroiwa@mail.utoronto.ca, jcb@mie.utoronto.ca

Abstract

Satisficing heuristic search such as greedy best-first search (GBFS) suffers from local minima, regions where heuristic values are inaccurate and a good node has a worse heuristic value than other nodes. Search algorithms that incorporate exploration mechanisms in GBFS empirically reduce the search effort to solve difficult problems. Although some of these methods entirely ignore the guidance of a heuristic during their exploration phase, intuitively, a good heuristic should have some bound on its inaccuracy, and exploration mechanisms should exploit this bound. In this paper, we theoretically analyze what a good node is for satisficing heuristic search algorithms and show that the heuristic value of a good node has an upper bound if a heuristic satisfies a certain property. Then, we propose biased exploration mechanisms which select lower heuristic values with higher probabilities. In the experiments using synthetic graph search problems and classical planning benchmarks, we show that the biased exploration mechanisms can be useful. In particular, one of our methods, Softmin-Type(h), significantly outperforms other GBFS variants in classical planning and improves the performance of Type-LAMA, a state-of-the-art classical planner.

Introduction

Greedy best-first search (GBFS) (Doran and Michie 1966) is a satisficing heuristic search algorithm widely used to solve difficult graph search problems including AI planning. GBFS estimates the distance to a goal node using a heuristic value and greedily expands nodes in the order of their heuristic values. However, GBFS suffers from local minima, regions where the heuristic values are inaccurate or misleading. In local minima, a good node, i.e., one that makes progress toward the goal node, has a higher heuristic value than other nodes, and GBFS needs to search through many nodes before reaching a good node. To escape local minima, a number of exploration methods have been developed (Imai and Kishimoto 2011; Valenzano et al. 2014; Xie, Müller, and Holte 2014; Xie et al. 2014). Usually, these algorithms perform GBFS most of the time but sometimes switch to exploration and select a node based on different criteria. Some of the algorithms such as ϵ-GBFS (Valenzano et al. 2014) and Type-GBFS (Xie et al. 2014), which empirically perform well, ignore the heuristic values during their exploration phase. However, although heuristics sometimes make mistakes, good heuristics should not be too inaccurate; intuitively, there should be some bound on the inaccuracy of a heuristic and exploration should consider this bound instead of completely ignoring heuristic values.

In this paper, we investigate exploration mechanisms that consider heuristic values. First, we theoretically analyze what a good node is for satisficing heuristic search algorithms. We prove that expanding *closest nodes*, the nodes closest to a goal node in the open list, is a sufficient and necessary conditions for open list-based search algorithms such as GBFS to find a solution. Then, we introduce the notion of t-*dominance*, a property, parameterized by function t, that measures the informativeness of a heuristic in a given graph search problem. We show that if a heuristic is t-dominating, the heuristic values of closest nodes have an upper bound depending on t, and we do not need to explore higher heuristic values. Although identifying t before solving a problem is impractical, from empirical observations, we expect that a good heuristic is usually t-dominating with some function t. Based on this idea, we propose biased exploration mechanisms, which select lower heuristic values with higher probabilities. In our experiments, we evaluate extensions of Type-GBFS with the biased exploration mechanisms. Using synthetic graph search problems, we validate that the biased exploration mechanisms can be useful when a heuristic is t-dominating. We also evaluate our methods in classical planning and show that Softmin-Type(h), one of the biased exploration mechanisms, significantly outperforms GBFS and Type-GBFS. Furthermore, we show that our method improves Type-LAMA (Xie et al. 2014), one of the state-of-the-art classical planners.

Satisficing Heuristic Search

We consider heuristic search algorithms that find a path in a directed graph. A *directed graph* is represented as $\langle V, E\rangle$ where V is a set of nodes and $E \subseteq V \times V$ is a set of edges. A *path* from node n_0 to node n_m is a sequence of edges $\langle e_1, ..., e_m\rangle$ where $\forall 1 \leq i \leq m, e_i = (n_{i-1}, n_i) \in E$. We say that node n is on the path if $n = n_i$ for some $0 \leq i \leq m$. Node n' is reachable from node n if there exists a path from n to n'. We call nodes which are reachable from node n *descendants* of n.

A *graph search problem* is defined as $\langle V, E, n_I, G\rangle$, where $\langle V, E\rangle$ is a directed graph, $n_I \in V$ is the *initial node*, and $G \subseteq V$ is the set of *goal nodes*. A solution for a graph search problem is a path from the initial node to a goal node in G, which is called a *solution path*. We denote the shortest path length from node n to a goal node by $d^*(n)$, the d^*-value of n. If no goal node is reachable from n, $d^*(n) = \infty$. We assume that a graph search problem is solvable, i.e., $d^*(n_I) < \infty$.

An algorithm which returns a solution for a graph search problem is called a *graph search algorithm*. We consider *open list algorithms*, graph search algorithms that maintain an *open list* $B \subseteq V$, which is the set of candidate nodes to search. At the beginning, $B = \{n_I\}$. At each step, an open list algorithm removes one node n from B and generates a *successor node* n' for each $(n, n') \in E$. From the generated successor nodes, some nodes can be pruned; for example, some search algorithms maintain a *closed list*, the set of already expanded nodes, and prune a node if it is included in the closed list. After pruning, the algorithm inserts the remaining successors into the open list. We call this step the *expansion* of n or say that the algorithm *expands* n. The algorithm terminates when it expands a goal node and constructs a path from n_I to the goal node by traversing the edges backwards. By $g(n)$, we denote the length of the path from n_I to n found by a search algorithm, the g-value of n. If there are multiple paths to n, we assume that a search algorithm chooses one of them.

Heuristic search algorithms use *heuristic* $h : V \to \mathbb{R}_0^+$, which estimates $d^*(n)$ by $h(n)$, the *h-value (heuristic value)* of n. Greedy best-first search (GBFS) (Doran and Michie 1966) expands a node having the minimum h-value from an open list and prunes successors using a closed list. When there are multiple nodes with the same h-value, GBFS selects one of them according to a *tie-breaking* strategy. For example, the first-in-first-out (FIFO) tie-breaking strategy expands the oldest node. There are variants of GBFS using *exploration mechanisms*, which expand a node based on different criteria from GBFS. ϵ-GBFS (Valenzano et al. 2014) performs GBFS with the probability of $1 - \epsilon$ and expands a node uniformly at random in the open list with the probability of ϵ at each expansion. Type-GBFS (Xie et al. 2014) alternately expands a node from two open lists B^1 and B^2 while using a shared closed list. A generated node is inserted to both of B^1 and B^2 if it is not in the closed list. When a node selected for expansion is already in the closed list, it is removed from the open list and not expanded. From B^1, a node having the minimum h-value is expanded as in GBFS. In B^2, node n is associated with type $T(n)$. First, type $\hat{T}$ is selected uniformly at random from $T(B^2) = \{T(n) \mid n \in B^2\}$, and then a node is expanded uniformly at random from $B^2_{\hat{T}} = \{n \in B^2 \mid T(n) = \hat{T}\}$. Empirically, $T(n) = (h(n), g(n))$, a tuple of the h-value and the g-value, works well.

Theoretical Analysis

Since GBFS always expands a node having the minimum h-value, if a good node is mistakenly assigned a high h-value, GBFS expands many nodes before expanding the good node. Exploration mechanisms such as ϵ-GBFS (Valenzano et al. 2014) and Type-GBFS (Xie et al. 2014) can be useful in such a case because they expand diverse nodes, which may have higher h-values than the minimum h-value. However, we lack a formal definition of a good node. Previous work theoretically analyzed the search behavior of GBFS and defined the notion of the bench transition system (Heusner, Keller, and Helmert 2017). In their analysis, once GBFS expands node n called a bench-exit state, all nodes expanded after n are descendants of n. Therefore, GBFS makes progress when it expands a bench-exit state, and a good node for GBFS is a bench-exit state. However, this theoretical analysis is limited to GBFS; for other algorithms, there is no guarantee that they expand only descendants of a bench-exit state, and expanding a bench-exit state does not necessarily mean making progress. Therefore, we introduce the notion of a good node for any open list algorithm. Our intuition is that the algorithm makes progress when it expands a node in the open list which is the closest to a goal node.

Definition 1 (Closest nodes)**.** *Given graph search problem $\langle V, E, n_I, G\rangle$ and open list $B \subseteq V$, $n' \in B$ is a closest node if $d^*(n') = \min_{n \in B} d^*(n)$.*

In fact, to find a solution, an open list algorithm must expand as many closest nodes as the length of the shortest solution path.

Theorem 1. *Given graph search problem $\langle V, E, n_I, G\rangle$, if an open list algorithm returns a solution path, it expands at least $d^*(n_I) + 1$ closest nodes.*

Proof. The minimum d^*-value of a node in the open list is decreased only when new nodes are added to the open list, i.e., a node is expanded and at least one of its successors is not pruned. Let the open list be B. Suppose that $n \in B$ is expanded, the open list is updated to B', and the minimum d^*-value is decreased. Since the minimum d^*-value is decreased, B' contains new node n' with $d^*(n') < \min_{n'' \in B} d^*(n'')$, which is a successor of n. d^*-values are integer, so $d^*(n') + 1 \leq \min_{n'' \in B} d^*(n'')$. Since $n \in B$, $d^*(n') + 1 \leq d^*(n)$. Since n' is a successor of n, $d^*(n') \geq d^*(n) - 1$. Therefore, $d^*(n) = d^*(n') + 1 \leq \min_{n'' \in B} d^*(n'')$, so n is a closest node in B. Thus, when the minimum d^*-value of a node in the open list is decreased, a closest node is expanded.

At the beginning, the minimum d^*-value is $d^*(n_I)$. When a goal node is in the open list, the minimum d^*-value is 0. A goal node is also a closest node. Therefore, when a goal node is expanded, at least $d^*(n_I) + 1$ closest nodes have been expanded. □

If an open list algorithm does not prune a successor node which will be a new closest node, it expands exactly $d^*(n_I) + 1$ closest nodes to find a solution.

Theorem 2. *Given graph search problem $\langle V, E, n_I, G\rangle$, suppose that when an open list algorithm expands a node from open list B, it does not prune at least one successor n' such that $d^*(n') < \min_{n'' \in B} d^*(n'')$. The algorithm finds a solution path iff it expands $d^*(n_I) + 1$ closest nodes.*

Proof. Let the open list be B. Suppose that $0 < \min_{n'' \in B} d^*(n'') < \infty$, $n \in B$ is expanded, and the open list is updated to B'. Since n is not a goal node, there exists at least one path from n to a goal node with the length of $d^*(n)$. One successor n' of n is on this path, and $d^*(n') = d^*(n) - 1$.

If n is a closest node, $d^*(n') < d^*(n) \leq \min_{n'' \in B} d^*(n'')$. n' is not pruned since it satisfies the condition of the theorem, $d^*(n') < \min_{n'' \in B} d^*(n'')$. n' becomes a closest node in B', and the minimum d^*-value of a node in the open list is decreased by 1.

If n is not a closest node, $d^*(n) > \min_{n'' \in B} d^*(n'')$. The d^*-value of every successor is at least $d^*(n) - 1 \geq \min_{n'' \in B} d^*(n'')$. Therefore, no matter which nodes are pruned or inserted to the open list, the minimum d^*-value is not changed.

Thus, when expanded node n is not a goal node, the minimum d^*-value of a node in the open list is not increased, and it is decreased by 1 if n is a closest node. At the beginning, the minimum d^*-value is $d^*(n_I)$. After expanding $d^*(n_I)$ closest nodes, the minimum d^*-value becomes 0, which means that closest nodes in the open list are goal nodes. Therefore, when the algorithm expands one more closest node, it finds a solution path. □

Expanded closest nodes are not necessarily on a solution path except for the initial and goal nodes. For example, after an algorithm expands closest node n with $d^*(n) = 1$, it may find a solution path which does not include n and nodes expanded before n but for n_I. In this case, the algorithm is making a detour despite the fact that a goal node, which is a successor of n, is already included in the open list.

If an open list algorithm prunes successors using only a closed list, the condition in Theorem 2 is satisfied. Therefore, GBFS, ϵ-GBFS, and Type-GBFS expand exactly $d^*(n_I) + 1$ closest nodes.

Theorem 3. *Given graph search problem $\langle V, E, n_I, G \rangle$, suppose that an open list algorithm prunes a successor using only a closed list. The algorithm finds a solution path iff it expands $d^*(n_I) + 1$ closest nodes.*

Proof. First, we show that for each expanded node n_0, at least one node on a shortest path from n_0 to a goal node is included in the open list if $d^*(n_0) < \infty$ and n_0 is not a goal node. Let a shortest path from n_0 to a goal node be $\langle e_1, ..., e_{d^*(n)} \rangle$ where $e_i = (n_{i-1}, n_i)$ for $1 \leq i \leq d^*(n)$ and $n_{d^*(n)} \in G$. Let j be the maximum i such that $0 \leq i \leq d^*(n)$ and n_i was expanded. Such a j exists since at least n_0 was expanded. $j \leq d^*(n) - 1$ since a goal node is not expanded yet. n_{j+1} is not expanded by the definition of j. Since n_{j+1} is a successor of an expanded node, it was not pruned and is inserted to the open list.

Suppose that node n is expanded from open list B and its successor n' is pruned. Since it is pruned, n' was already expanded. If $d^*(n') = \infty$, then $d^*(n') \geq \min_{n'' \in B} d^*(n'')$. Otherwise, there exists node $\hat{n}$ on a shortest path from n' to a goal node in the open list. Since $d^*(n') > d^*(\hat{n})$, $d^*(n') \geq \min_{n'' \in B} d^*(n'')$. Thus, the pruning satisfies the condition in Theorem 2. □

Based on Theorem 3, for GBFS and its variants, we assume that expanding a closest node means making progress, and a closest node is a good node. Therefore, an exploration mechanism that quickly expands a closest node is desirable.

Heuristic Values and Closest Nodes

Based on the notion of closest nodes, we investigate when exploration mechanisms are needed and what kind of exploration mechanisms are efficient. To identify closest nodes, we need to know d^*-values of all nodes in the open list. By definition of d^*-values, we need to compute the shortest paths from all nodes in the open list to a goal node, which is at least as difficult as solving the graph search problem. Therefore, we estimate d^*-values from information available without expensive computation. In satisficing heuristic search, h-values can be considered such estimation of d^*-values. If the estimation is perfect, we do not need to use exploration mechanisms; if $\forall n \in V, h(n) = d^*(n)$, GBFS immediately finds a solution. In fact, even if $h(n) \neq d^*(n)$, GBFS immediately finds a solution as long as the order of h-values is consistent with the order of d^*-values. We formalize this property of a satisficing heuristic.

Definition 2 (Perfect satisficing heuristic)**.** *Given graph search problem $\langle V, E, n_I, G \rangle$, heuristic h is a perfect satisficing heuristic if*

$$\forall n_1, n_2 \in V, h(n_1) \leq h(n_2) \rightarrow d^*(n_1) \leq d^*(n_2).$$

Theorem 4. *Given graph search problem $\langle V, E, n_I, G \rangle$ and heuristic h, GBFS expands a closest node from any open list $B \subseteq V$ regardless of the tie-breaking strategy iff h is a perfect satisficing heuristic.*

Proof. Let n be the node expanded by GBFS. $\forall n' \in B, d^*(n) \leq d^*(n')$ since $h(n) \leq h(n')$. Therefore, $d^*(n) = \min_{n' \in B} d^*(n')$, so n is a closest node.

Suppose $B = \{n_1, n_2\}$ for any pair of nodes n_1 and n_2 with $d^*(n_2) < d^*(n_1)$ in V. If GBFS expands a closest node for any open list regardless of the tie-breaking strategy, since GBFS expands n_2 from B, $h(n_2) < h(n_1)$. Therefore,

$$\forall n_1, n_2 \in V, d^*(n_2) < d^*(n_1) \rightarrow h(n_2) < h(n_1),$$

which is the contrapositive of the definition of a perfect satisficing heuristic. □

From Theorems 3 and 4, GBFS is an optimal algorithm with a perfect satisficing heuristic.

Corollary 1. *Given graph search problem $\langle V, E, n_I, G \rangle$ and a perfect satisficing heuristic h, GBFS minimizes the number of expansions to find a solution.*

Theorem 4 says that using a perfect satisficing heuristic is a sufficient and necessary condition for GBFS to always expand a closest node *given any subset of V as an open list*. However, in practice, GBFS never encounters some subsets of V, so GBFS can be an optimal algorithm without a perfect satisficing heuristic. For example, if all nodes on one shortest solution path have h-values of 0 and other nodes have h-values of ∞, GBFS expands a closest node at every

expansion. The heuristic is not a perfect satisficing heuristic because for n_1 and its successor n_2 on a shortest path, $d^*(n_2) < d^*(n_1)$ and $h(n_2) = h(n_1)$. This does not contradict Theorem 4; given open list $\{n_1, n_2\}$, GBFS may expand n_1, which is not a closest node. However, since n_2 is a successor of n_1, GBFS never encounters such an open list.

t-Dominance

Since the exploration mechanisms empirically work well, we do not assume that actual heuristics are perfect satisficing. However, if $h(n_1)$ is sufficiently lower than $h(n_2)$, we expect that n_1 has lower d^*-value than n_2. We formalize this intuition as the following property of a heuristic.

Definition 3 (t-dominance)**.** *Given graph search problem $\langle V, E, n_I, G\rangle$, heuristic h, and function $t : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ such that $t(x) \geq x$, h is a t-dominating heuristic if*

$$\forall n_1, n_2 \in V, t(h(n_1)) < h(n_2) \to d^*(n_1) < d^*(n_2).$$

Note that $t(x) \geq x$ is necessary since $t(x) < x$ causes a contradiction when $n_1 = n_2$. With a t-dominating heuristic, we can infer a bound on h-values of closest nodes based on t. Intuitively, nodes having h-values higher than some relative threshold cannot be closest nodes, so an exploration mechanism will not benefit from expanding such nodes.

Proposition 1. *Given graph search problem $\langle V, E, n_I, G\rangle$, open list $B \subseteq V$, and t-dominating heuristic h, any node $n' \in B$ with $h(n') > \min_{n \in B} t(h(n))$ is not a closest node.*

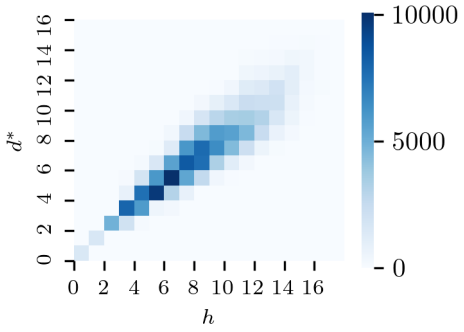
There is no guarantee that nodes having lower h-values than the threshold are closest nodes since the heuristic might not be a perfect satisficing heuristic; even when t is an identity function, it is still possible that nodes n_1 and n_2 with $h(n_1) = h(n_2)$ satisfy $d^*(n_1) > d^*(n_2)$, which does not violate the condition of t-dominance, $h(n_1) < h(n_2) \to d^*(n_1) < d^*(n_2)$, but violates the condition of a perfect satisficing heuristic, $h(n_1) \leq h(n_2) \to d^*(n_1) \leq d^*(n_2)$.

As an example of t-dominance, we present a t-function for the unit-cost version FF heuristic (Hoffmann and Nebel 2001) in a classical planning problem instance, ptesting-2-2-3 of the HIKING domain from the optimal track of IPC14. We show the number of nodes having a particular combination of an h-value and a d^*-value using a heatmap in Figure 1. We also show the t-function empirically computed based on h-values and d^*-values of all nodes; for h-value x, $t(x)$ is the maximum h-value of node n in V such that $d^*(n) \leq \max_{n' \in V : h(n') = x} d^*(n')$. If $h(n'') > t(x)$ for node n'', $d^*(n'') > \max_{n' \in V : h(n') = x} d^*(n')$, so the condition of a t-dominating heuristic is satisfied. In the example, when $x \leq 7$, $t(x)$ is less than the maximum h-value, 17. In other words, when $h(n) \leq 7$, nodes having higher h-values than some relative thresholds always have higher d^*-values than $d^*(n)$. As t-dominance is parameterized by a function, not a single value, quantitatively analyzing t-dominance using a set of problems is difficult. However, we observe qualitatively similar tendencies in instances in many other classical planning domains.

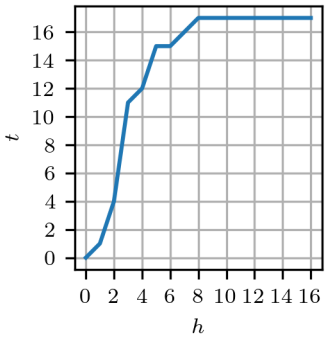
Summary

In this section, we showed that open list algorithms including GBFS and its variants must expand closest nodes to find a solution path, and therefore expanding a closest node can be considered as making progress for such algorithms (Theorem 1–3). Then, we showed that with a perfect satisficing heuristic, GBFS always expands a closest node, and we do not need exploration mechanisms (Theorem 4 and Corollary 1). Since we do not expect a heuristic is perfect satisficing in practice, we introduced the notion of t-dominance, which parameterizes the informativeness of a heuristic using a heuristic- and problem-specific function t. We showed that closest nodes have lower h-values than a threshold depending on t when a heuristic is t-dominating (Proposition 1).

If we knew a t-function with which a heuristic is t-dominating, exploration mechanisms could avoid nodes having higher h-values than the threshold. Unfortunately, identifying a t-function requires computing d^*-values of all nodes. However, we expect that a good heuristic is t-dominating for some t, and closest nodes have lower h-values than some threshold. In the following section, we consider improving exploration mechanisms based on this expectation. Intuitively, we try to increase the probability to expand a closest node by biasing expansion to nodes having lower h-values.



(a) Distribution of d^*- and h-values.



(b) t function.

Figure 1: Example of t-dominance using the HIKING domain from IPC14 and the FF heuristic.

Improving Exploration Mechanisms

First, we formally define an exploration mechanism as a probability distribution over nodes in the open list.

Definition 4. *Given graph search problem $\langle V, E, n_I, G\rangle$ and open list $B \subseteq V$, $p : B \to \mathbb{R}_0^+$ is an exploration mechanism if $\sum_{n\in B} p(n) = 1$. When an open list algorithm uses p given B, it expands $n \in B$ with the probability of $p(n)$.*

For example, Type-GBFS alternately uses GBFS and exploration mechanism p_T such that $p_T(n) = \frac{1}{|T(B^2)||B^2_{T(n)}|}$ for $n \in B^2$. Although p_T expands all nodes with non-zero probabilities, we actually do not need to consider nodes having higher h-values than a certain threshold if the heuristic is t-dominating according to Proposition 1. One potential problem is that p_T can be biased toward non-closest nodes if there are many types with h-values higher than the threshold. In such cases, it expands a non-closest node with a high probability. The same problem is also possible for ϵ-GBFS if many nodes have higher h-values than the threshold. To avoid this bias, we propose a variant of Type-GBFS where first h-value $\hat{h}$ is selected uniformly at random from $H(B) = \{h(n) \mid n \in B\}$, then type $\hat{T}$ is selected uniformly at random from $T(B_{\hat{h}}) = \{T(n) \mid n \in B \wedge h(n) = \hat{h}\}$, and finally a node is expanded uniformly at random from $B_{\hat{h},\hat{T}} = \{n \in B \mid h(n) = \hat{h} \wedge T(n) = \hat{T}\}$.

Definition 5. *Given graph search problem $\langle V, E, n_I, G\rangle$ and open list $B \subseteq V$, exploration mechanism $p_{h,T}$ is defined as*

$$\forall n \in B, p_{h,T}(n) = \frac{1}{|H(B)||T(B_{h(n)})||B_{h(n),T(n)}|}.$$

Since all h-values are selected with the same probability, this exploration mechanism is not biased toward particular h-values with which many types are associated. However, if there are many h-values higher than the threshold, the exploration mechanism is still biased toward higher h-values.

Another approach to improve exploration mechanisms is to estimate the threshold by some value t' and avoid expanding nodes having higher h-values than t'. However, if $t' < \min_{n\in B} t(h(n))$, i.e., we underestimate, it is possible that all closest nodes have higher h-values than t'. In such a case, if an exploration mechanism focuses only on h-values less than or equal to t', it cannot find a closest node. In addition, expanding all nodes with non-zero probability is theoretically beneficial because it ensures the probabilistic completeness of a search algorithm for a graph search problem with an infinite graph (Valenzano and Xie 2016). Therefore, instead of using a hard threshold like t', we use a soft threshold; we use biased exploration mechanisms which expand all nodes with non-zero probabilities but assign higher probabilities to nodes with lower h-values.

Definition 6. *Given graph search problem $\langle V, E, n_I, G\rangle$, open list $B \subseteq V$, heuristic h, and exploration mechanism p, let $w : H(B) \to \mathbb{R}_0^+$ be a function such that $\sum_{n\in B} w(h(n))p(n) = 1$. Biased exploration mechanism $w[p]$ is defined as*

$$\forall n \in B, w[p](n) = w(h(n))p(n).$$

Proposition 2. *Given graph search problem $\langle V, E, n_I, G\rangle$, open list $B \subseteq V$, heuristic h, exploration mechanism p, and biased exploration mechanism $w[p]$, if h is a t-dominating heuristic and $w : H(B) \to \mathbb{R}_0^+$ is a non-increasing function,*

$$q(B, w[p]) \geq w\left(\min_{n\in B} t(h(n))\right) q(B, p).$$

where $q(B, p)$ is the probability that exploration mechanism p expands a closest node.

Proof. $q(B, p) = \sum_{n\in Y} p(n)$ and $q(B, w[p]) = \sum_{n\in Y} w(h(n))p(n)$ where Y is the set of closest nodes in B. Since h is t-dominating, $\forall n' \in Y, h(n') \leq \min_{n\in B} t(h(n))$. Since w is non-increasing, $\forall n' \in Y, w(h(n')) \geq w\left(\min_{n\in B} t(h(n))\right)$. □

The biased exploration mechanism expands a closest node with a higher probability than the original one if $w(\min_{n\in B} t(h(n))) > 1$, which is more likely to occur when $\min_{n\in B} t(h(n))$ is lower, i.e., a heuristic is more informative.

We propose the following non-increasing functions as w:

- $\mathrm{lin}(\hat{h}) = \frac{\sup H(B) - \alpha\hat{h} + \beta}{\sum_{n\in B}(\sup H(B) - \alpha h(n) + \beta)p(n)}$ where $\alpha \geq 0$ and $\beta \geq 1$.
- $\mathrm{softmin}(\hat{h}) = \frac{\exp(-\hat{h}/\tau)}{\sum_{n\in B} \exp(-h(n)/\tau)p(n)}$ where $\tau > 0$.

With lin, the probability to expand a node decreases linearly with higher h-value. Parameters α and β are the weight and the bias of the linear function. $\beta \geq 1$ guarantees non-negativity. The probabilities to expand nodes having lower h-values increase as α increases and β decreases. With softmin, the probability to expand a node decreases exponentially with higher h-value. The probabilities to expand nodes having lower h-values increases as τ decreases. In these functions, each value is divided by the sum of values over all nodes so that $\sum_{n\in B} w(h(n))p(n) = 1$.

Experimental Evaluation

We compare the following algorithms with GBFS:

- Type: Type-GBFS where $T(n) = (h(n), g(n))$.
- Type(h): Type using $p_{h,T}$ as the exploration mechanism instead of p_T.
- 3-Type(h): Type(h) where the exploration mechanism ignores nodes having h-values higher than the third lowest h-value in the open list.
- Lin-Type(h): Type(h) using $\mathrm{lin}[p_{h,T}]$ with $\alpha = 1$ and $\beta = 1$ as the exploration mechanism instead of $p_{h,T}$.
- Softmin-Type(h): Type(h) using $\mathrm{softmin}[p_{h,T}]$ with $\tau = 1$ as the exploration mechanism instead of $p_{h,T}$.

Here, we focus on extending Type-GBFS since it is shown to be better than ϵ-GBFS by previous work (Xie et al. 2014). For 3-Type(h), Lin-Type(h) and Softmin-Type(h), we do not tune the parameters and use fixed values for all experiments. Optimization of the parameters is future work. As Softmin-Type(h) uses an exponential function for weighting, it is expected to be greedier than Lin-Type(h). By comparing 3-Type(h) to Lin-Type(h) and Softmin-Type(h), we evaluate the benefit of using a non-decreasing function instead of a hard threshold. We use FIFO tie-breaking strategy for GBFS and the first open list of Type-GBFS and its variants.

Synthetic Search Space

First, we verify that biased exploration mechanisms are useful when a heuristic is t-dominating. We generate synthetic graph search problems and assign heuristic values so that the heuristic is t-dominating. We evaluate the exploration mechanisms using heuristics with different t functions. Following previous work which empirically analyzed GBFS using a synthetic search space (Cohen and Beck 2017), we use $D_{m,p}$, the probability space of all random directed graphs with m nodes and no self-loops where each directed edge is present with probability p (Karp 1990). We generate graph search problem $\langle V, E, n_I, G\rangle$ in the following procedure:

1. Sample directed graph $\langle V, E\rangle$ from $D_{m,p}$.
2. Randomly choose node $n_g \in V$ such that $\exists n, (n, n_g) \in E$. Let n_g be a goal node, i.e., $G = \{n_g\}$.
3. Randomly choose initial node $n_I \neq n_g$ such that n_g is reachable from n_I.

Differently from the previous work by Cohen and Beck, a generated problem is always solvable since the goal node is reachable from the initial node. We use a t-dominating heuristic such that search algorithms encounter local minima. Since we are focusing on exploration mechanisms that are different in the selection of an h-value, we want to reduce the effect of a tie-breaking strategy. We design the heuristic so that it assigns different h-values to nodes having different d^*-values. Our heuristic is defined as follows:

$$h(n) = \begin{cases} \infty & \text{if } d^*(n) = \infty \\ 0 & \text{if } d^*(n) = 0 \\ d^*(n) + \delta & \text{if } d^*(n) \equiv 1 \mod (\delta + 1) \\ d^*(n) - 1 & \text{otherwise} \end{cases} \quad (1)$$

where $\delta \geq 1$ is an integer parameter. When n is a closest node and $h(n) = d^*(n) + \delta$, GBFS needs to expand each node n' with $d^*(n) < d^*(n') \leq d^*(n) + \delta$ before n since $h(n') < h(n)$. As δ increases, the number of such nodes is expected to increase. This heuristic is t-dominating with $t(h(n)) = h(n) + \delta$; we show that $d^*(n_1) < d^*(n_2)$ for nodes n_1 and n_2 with $h(n_1) + \delta < h(n_2)$. Since $d^*(n) - 1 \leq h(n) \leq d^*(n) + \delta$ for node n, $d^*(n_1) - 1 + \delta < d^*(n_2) + \delta$, so $d^*(n_1) < d^*(n_2) + 1$. If $d^*(n_1) = d^*(n_2)$, $h(n_1) = h(n_2)$, which contradicts that $h(n_1) + \delta < h(n_2)$. Therefore, $d^*(n_1) < d^*(n_2)$.

In the experiment, following Cohen and Beck, we generate graph search problems with $m = 10000$ and $p = \frac{2}{m-1}$ considering only instances having at least 1000 edges.[1] For the heuristic, we use $\delta \in \{1, ..., 9\}$. For each value of δ, we generate 1000 instances and evaluate the median number of expansions to solve a problem. In addition to the algorithms presented above, we also evaluate δ-Type(h) which is Type(h) ignoring nodes having h-values higher than $\min_{n \in B} h(n) + \delta$ given open list B and parameter δ. In other words, δ-Type(h) uses an exploration mechanism which exploits perfect information of the t function. We implement the problem generation method and search algorithms in Python 3.8.10 and run the experiment on a machine having Intel Core i5-8265U and 16GB RAM.

[1] $p = \frac{2}{m-1}$ corresponds to $\gamma = 2$ in Cohen and Beck (2017).

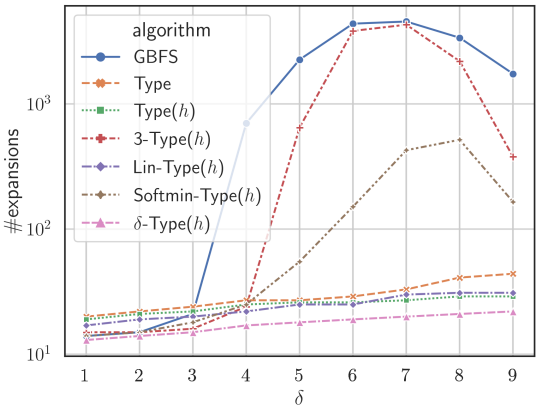


Figure 2: Search effort to solve a synthetic graph search problem using the heuristic in Equation 1 with different δ.

We show the result in Figure 2. As expected, δ-Type(h), which exploits the perfect information, is the best algorithm for all values of δ. For $\delta \geq 4$, Type and Type(h) expand fewer nodes than GBFS, which is consistent with the intuition that exploration mechanisms are more useful when a heuristic is less informative. Type(h) is better than Type for all values of δ. The biased exploration mechanisms, 3-Type(h), Lin-Type(h), and Softmin-Type(h), are better than Type(h) when δ is small. In particular, when $\delta = 3$, they expand fewer nodes than GBFS and Type(h), which confirms that biased exploration mechanisms are helpful when a heuristic is informative to some extent. However, as δ increases, the biased exploration mechanisms expand more nodes than Type(h). 3-Type(h) is similar to Softmin-Type(h), but it expands almost 10 times more nodes when δ is large. Lin-Type(h) is more exploratory and similar to Type(h) as the non-increasing weight function is not as steep as 3-Type(h) and Softmin-Type(h).

When we change the value of p, the tendency is qualitatively similar. Increasing p, we observe that GBFS expands more than 1000 nodes even when $\delta \leq 3$ possibly because more nodes become reachable from a node, and GBFS needs to expand them before a closest node.

Classical Planning

We evaluate the exploration mechanisms in classical planning. We use the Autoscale benchmark set 21.08, the recently developed benchmark set of classical planning instances (Torralba, Seipp, and Sievers 2021). In version 21.08, some instances in PARCPRINTER and PATHWAYS are duplicates. In domains that are not unit-cost, we compute h-values and g-values ignoring action costs; we use the unit-cost version of the FF heuristic (Hoffmann and Nebel 2001) with eager evaluation for the heuristic function, and the g-value of node n is the length of the path from the initial node to n. We implement all the methods in the Fast Downward planning system (Helmert 2006). We run the experiment using an Intel Xeon Gold 6148 processor with a 30-minutes

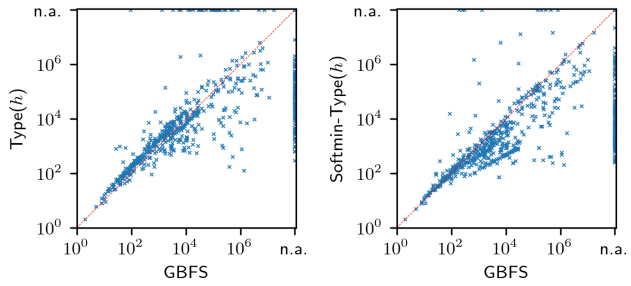


Figure 3: Comparison of the number of expansions of GBFS vs. GBFS with exploration mechanisms. For Type(h) and Softmin-Type(h), we use the results of one run. Unsolved instances are shown at 'n.a.'.

time limit and 4 GB memory limit for each instance using GNU parallel (Tange 2011).

We show the coverage of the algorithms in the left-hand side of Table 1. Except for GBFS, which is deterministic, we show the average over five runs. Type(h) solves more instances than Type in total. Comparing the biased exploration mechanisms, Softmin-Type(h) is the best and 3-Type(h) is the next. Overall, Softmin-Type(h) solves more instances than GBFS, Type, and Type(h) in 22 out of 41 domains. It maintains similar coverage to GBFS in domains where Type and Type(h) degrade such as BARMAN, ELEVATORS, and GED while benefiting from exploration in domains where Type and Type(h) perform better such as PIPESWORLD, TIDYBOT, and VISITALL. These results suggest that the FF heuristic is so informative that the aggressively biased exploration mechanisms such as Softmin-Type(h) are useful.

We compare the number of expansions of GBFS, Type(h), and Softmin-Type(h) for each instance in Figure 3. Although Type(h) significantly reduces the number of expansions in hard instances, where GBFS expands more than 100000 nodes, it tends to expand more nodes in easy instances. In contrast, Softmin-Type(h) behaves similarly to GBFS in easy instances while it reduces the search effort in hard instances similarly to Type(h). Compared to Type(h), Softmin-Type(h) achieves the better balance between exploration and exploitation.

While a node is removed in a constant time in the original type-based open list, a biased exploration mechanism requires computational time proportional to the number of h-values in the open list since it computes an weight for each h-value. However, we do not observe major computational overhead of biased exploration mechanisms in practice.

We use the biased exploration mechanisms in Type-LAMA (Xie et al. 2014), which incorporates the type-based open list in LAMA (Richter and Westphal 2010). Type-LAMA is used in state-of-the-art classical planner portfolios (Seipp 2018; Seipp and Röger 2018). We evaluate the following variants of LAMA in the same resource setting as the evaluation of GBFS and its variants in classical planning.

1. LAMA (Richter and Westphal 2010).
2. Type-FF: original Type-LAMA (Xie et al. 2014), which is LAMA with an additional type-based open list with exploration mechanism p_T and type $T(n) = (h^{\mathrm{FF}}(n), g(n))$ where $h^{\mathrm{FF}}(n)$ is the h-value of node n computed by the FF heuristic.
3. Type-FF-LC: Type-FF with an additional type-based open list with exploration mechanism p_T and type $T(n) = (h^{\mathrm{LC}}(n), g(n))$ where $h^{\mathrm{LC}}(n)$ is the h-value of n by the landmark count heuristic (Richter, Helmert, and Westphal 2008; Richter and Westphal 2010).
4. SM-FF: Type-FF with exploration mechanism softmin$[p_{h,T}]$ instead of p_T.
5. SM-FF-LC: Type-FF-LC with exploration mechanism softmin$[p_{h,T}]$ instead of p_T.

We show the coverage in the right-hand side of Table 1. Except for LAMA, which is deterministic, we show the average over five runs. Using softmin$[p_{h,T}]$ improves the performance of both of Type-FF and Type-FF-LC, and SM-FF-LC achieves the best coverage. While Type-FF-LC, where two out of four open lists are type-based, performs worse than Type-FF, SM-FF-LC is better than SM-FF. This result suggests that having two original type-based open lists is too explorative, but it results in a good balance with softmin$[p_{h,T}]$.

Related Work

Previous work theoretically and empirically analyzed the behavior of satisficing heuristic search algorithms and local minima of a graph search problem. For example, Hoffmann (2005) analyzed the behavior of enforced hill climbing with the FF heuristic (Hoffmann and Nebel 2001) in classical planning. Wilt and Ruml (2014; 2015) investigated which types of heuristics are effective for GBFS to escape local minima. Heusner, Keller, and Helmert (2017) theoretically identified when GBFS makes progress. As we mentioned above, the notion of bench-exit states in their work is similar to closest nodes but limited to GBFS. Cohen and Beck (2018a; 2018b) empirically showed that the largest search effort to escape a single local minimum is highly correlated with the overall search effort to solve a problem and that the existence of a deep local minimum is related to the heavy-tailed distribution of search effort. None of the previous work defined a good node independently from particular heuristic search algorithms.

A number of GBFS variants have been proposed to escape local minima. ϵ-GBFS (Valenzano et al. 2014) and Type-GBFS (Xie et al. 2014) use randomized exploration mechanisms ignoring the order of heuristic values. DBFS (Imai and Kishimoto 2011) and GBFS-LE (Xie, Müller, and Holte 2014) perform local search using a separate open list. Although DBFS is a complicated algorithm, it is similar to our biased exploration mechanisms in that it selects a node, from which local search is performed, according to the probability biased by its heuristic value. RR-GBFS (Cohen and Beck 2018b) uses randomized restarts to avoid getting stuck in a large local minimum. Asai and Fukunaga (2017) proposed a randomized tie-breaking strategy for GBFS to explore diverse nodes. Since these two methods are orthogonal to Type-GBFS, they can be combined with our biased exploration mechanisms.

domain	GBFS	Type	Type(h)	3	Lin	SM	LAMA	Type-FF	Type-FF-LC	SM-FF	SM-FF-LC
AGRICOLA (30)	17	19.6	**27.0**	17.2	20.2	18.0	21	**26.4**	22.4	23.8	25.8
AIRPORT (30)	16	15.8	16.0	**21.0**	16.8	18.8	14	15.8	**16.8**	14.8	14.2
BARMAN (30)	12	6.4	6.0	**12.8**	7.6	12.6	30	30.0	30.0	30.0	30.0
BLOCKSWORLD (30)	5	6.0	6.0	**7.8**	7.6	**7.8**	**27**	**27.0**	26.8	26.8	26.8
CHILDSNACK (30)	**5**	2.8	3.0	1.6	3.4	3.4	9	9.0	9.0	9.0	9.0
DATA-NETWORK (30)	5	8.4	7.0	**12.8**	10.6	12.4	20	20.4	19.6	**22.6**	21.6
DEPOTS (30)	9	9.0	9.0	9.2	9.0	**9.6**	**21**	**21.0**	20.8	20.2	20.2
DRIVERLOG (30)	7	7.2	8.0	8.2	8.2	**10.0**	29	**29.2**	29.0	29.0	29.0
ELEVATORS (30)	28	23.0	25.0	**30.0**	24.4	**30.0**	30	30.0	30.0	30.0	30.0
FLOORTILE (30)	2	2.0	2.0	2.0	2.0	2.0	2	2.0	2.0	2.0	2.0
FREECELL (30)	4	4.0	4.0	4.0	4.0	4.0	4	4.0	4.0	4.0	4.0
GED (30)	26	8.6	11.0	**28.8**	10.8	28.2	30	30.0	30.0	30.0	30.0
GRID (30)	8	8.4	8.0	9.0	8.6	**9.4**	18	18.0	18.0	18.0	18.0
GRIPPER (30)	30	30.0	30.0	30.0	30.0	30.0	30	30.0	30.0	30.0	30.0
HIKING (30)	18	13.0	16.0	16.6	15.8	**19.4**	**17**	9.2	6.8	13.8	10.8
LOGISTICS (30)	9	8.0	8.0	11.2	8.2	**13.0**	**11**	9.0	9.0	**11.0**	9.4
MICONIC (30)	30	30.0	30.0	30.0	30.0	30.0	30	30.0	30.0	30.0	30.0
NOMYSTERY (30)	9	13.0	15.0	6.4	**15.2**	7.2	**17**	16.8	16.8	16.4	15.0
OPENSTACKS (30)	9	9.0	9.0	15.0	10.4	**18.0**	30	30.0	30.0	30.0	30.0
ORGANIC-SYNTHESIS-SPLIT (30)	**17**	16.4	**17.0**	15.2	16.8	15.4	17	18.0	**19.4**	17.0	18.4
PARCPRINTER (30)	**30**	**30.0**	**30.0**	29.6	**30.0**	**30.0**	30	30.0	30.0	30.0	30.0
PARKING (30)	8	6.4	6.0	**11.2**	6.2	10.8	22	22.0	22.0	22.0	22.0
PATHWAYS (30)	11	13.6	16.0	18.4	18.4	**18.8**	18	18.0	18.0	18.0	18.0
PEGSOL (30)	30	30.0	30.0	30.0	30.0	30.0	30	30.0	30.0	30.0	30.0
PIPESWORLD-NOTANKAGE (30)	11	23.8	**25.0**	21.2	23.6	23.2	23	24.2	**24.8**	23.2	24.0
PIPESWORLD-TANKAGE (30)	12	20.4	23.0	20.8	**23.6**	21.8	**30**	29.6	**30.0**	29.6	**30.0**
ROVERS (30)	15	14.2	16.0	**24.2**	16.2	23.4	30	30.0	30.0	30.0	30.0
SATELLITE (30)	**10**	**10.0**	9.0	9.8	9.4	**10.0**	18	18.0	18.0	18.0	18.0
SCANALYZER (30)	3	6.4	8.0	**9.4**	7.6	9.0	19	19.0	19.0	19.0	19.0
SNAKE (30)	9	8.0	**10.0**	8.4	8.0	9.0	9	9.6	10.4	9.0	**12.8**
SOKOBAN (30)	20	**23.6**	21.0	20.6	23.0	21.6	22	**23.0**	22.4	21.6	22.6
STORAGE (30)	3	6.2	7.0	**10.4**	7.6	10.0	11	9.8	9.8	**12.6**	12.0
TERMES (30)	17	17.0	15.0	**18.6**	17.2	**18.6**	**23**	22.2	22.0	22.6	**23.0**
TETRIS (30)	6	7.4	10.0	**10.8**	9.6	10.4	8	8.0	8.0	12.2	**13.4**
THOUGHTFUL (30)	12	**23.6**	23.0	9.8	22.6	11.0	19	23.0	**24.4**	19.4	19.2
TIDYBOT (30)	14	16.4	16.0	16.4	**17.6**	16.4	17	17.2	17.4	16.8	**17.8**
TPP (30)	7	8.2	9.0	10.0	9.2	**11.0**	20	20.0	20.0	20.0	20.0
TRANSPORT (30)	4	4.0	4.0	**5.8**	4.8	**5.8**	13	13.2	13.0	13.4	**14.2**
VISITALL (30)	5	**9.0**	8.0	6.2	8.6	7.0	30	30.0	30.0	30.0	30.0
WOODWORKING (30)	2	4.4	7.0	**9.0**	6.4	7.8	**11**	8.8	7.8	9.0	8.0
ZENOTRAVEL (30)	**11**	10.2	10.0	**11.0**	10.0	**11.0**	14	14.0	14.0	14.0	14.0
Total (1230)	506	533.4	560.0	600.4	569.2	**615.8**	824	825.4	821.4	828.8	**832.2**

Table 1: Coverage of classical planning benchmark instances. 3: 3-Type(h), Lin: Lin-Type(h), and SM: Softmin-Type(h).

In classical planning, best-first width search, which balances exploitation and exploration by combining the novelty and the heuristic value of a state in the priority function of best-first search, achieves the state-of-the-art performance (Lipovetzky and Geffner 2017). Fickert and Hoffmann (2017) proposed a different approach, which escapes local minima by refining a delete relaxation heuristic and changing heuristic values of nodes. Recent work has proposed a lookahead strategy for GBFS, which tries to find a node with a lower h-value than the expanded node at each expansion using both of novelty and delete relaxation heuristics (Fickert 2020). While the notions of novelty and delete relaxation heuristics used in these methods are limited to particular graph search problems such as classical planning, our approach does not have such a limitation.

Conclusion

In this paper, we introduced the notion of closest nodes and theoretically showed that expanding a closest node means making progress for open list-based satisficing heuristic search algorithms. Then, we showed that when a heuristic is t-dominating, closest nodes have lower heuristic values than some relative threshold and proposed the biased exploration mechanisms exploiting this property. Our methods are empirically more useful than conventional exploration mechanisms and improve a state-of-the-art classical planner.

In our experiments, parameters of the biased exploration mechanisms are not tuned. Tuning parameters to problem domains offline or online is a part of our future work. Since an exploration mechanism is a probability distribution, directly learning it or, equivalently, learning a t-function using machine learning techniques may also be possible.

Acknowledgments

Partial funding for this work was provided by the Natural Sciences and Engineering Research Council of Canada.

References

Asai, M.; and Fukunaga, A. 2017. Exploration Among and Within Plateaus in Greedy Best-First Search. In *Proc. ICAPS*, 11–19.

Cohen, E.; and Beck, J. C. 2017. Problem Difficulty and the Phase Transition in Heuristic Search. In *Proc. AAAI*, 780–786.

Cohen, E.; and Beck, J. C. 2018a. Fat- And Heavy-Tailed Behavior in Satisficing Planning. In *Proc. AAAI*, 6136–6143.

Cohen, E.; and Beck, J. C. 2018b. Local minima, heavy tails, and search effort for GBFS. In *Proc. IJCAI*, 4708–4714.

Doran, J. E.; and Michie, D. 1966. Experiments with the Graph Traverser program. *Proceedings of the Royal Society*, 294(1437): 235–259.

Fickert, M. 2020. A Novel Lookahead Strategy for Delete Relaxation Heuristics in Greedy Best-First Search. In *Proc. ICAPS*, 119–123.

Fickert, M.; and Hoffmann, J. 2017. Complete Local Search: Boosting Hill-Climbing through Online Relaxation Refinement. In *Proc. ICAPS*, 107–115.

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

Heusner, M.; Keller, T.; and Helmert, M. 2017. Understanding the search behaviour of greedy best-first search. In *Proc. SOCS*, 47–55.

Hoffmann, J. 2005. Where "Ignoring Delete Lists" Works: Local Search Topology in Planning Benchmarks. *JAIR*, 24: 685–758.

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

Imai, T.; and Kishimoto, A. 2011. A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning. In *Proc. SOCS*, 985–991.

Karp, R. M. 1990. The transitive closure of a random digraph. *Random Structures & Algorithms*, 1(1): 73–93.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *Proc. AAAI*, 3590–3596.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks Revisited. In *Proc. AAAI*, 975–982.

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

Seipp, J. 2018. Fast Downward Remix. In *IPC 2018 planner abstracts*, 80–82.

Seipp, J.; and Röger, G. 2018. Fast Downward Stone Soup 2018. In *IPC 2018 planner abstracts*, 74–76.

Tange, O. 2011. GNU Parallel - The Command-Line Power Tool. *;login: The USENIX Magazine*, 36: 42–47.

Torralba, Á.; Seipp, J.; and Sievers, S. 2021. Automatic Instance Generation for Classical Planning. In *Proc. ICAPS*, 376–384.

Valenzano, R.; Sturtevant, N. R.; Schaeffer, J.; and Xie, F. 2014. A Comparison of Knowledge-Based GBFS Enhancements and Knowledge-Free Exploration. In *Proc. ICAPS*, 375–379.

Valenzano, R.; and Xie, F. 2016. On the completeness of best-first search variants that use random exploration. In *Proc. AAAI*, 784–790.

Wilt, C.; and Ruml, W. 2014. Speedy Versus Greedy Search. In *Proc. SOCS*, 184–192.

Wilt, C.; and Ruml, W. 2015. Building a Heuristic for Greedy Search. In *Proc. SOCS*, 131–139.

Xie, F.; Müller, M.; and Holte, R. 2014. Adding Local Exploration to Greedy Best-First Search in Satisficing Planning. In *Proc. AAAI*, 2388–2394.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-Based Exploration with Multiple Search Queues for Satisficing Planning. In *Proc. AAAI*, 2395–2401.