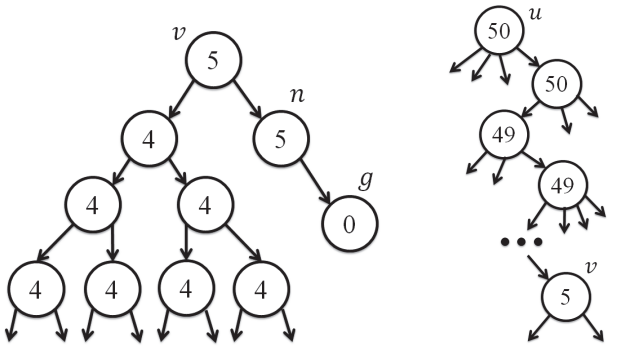
On the Completeness of Best-First Search Variants That Use Random Exploration

Richard Valenzano
University of Toronto
Toronto, Canada
rvalenzano@cs.toronto.edu

Fan Xie
University of Alberta
Edmonton, Canada
fxie2@ualberta.ca

Abstract

While suboptimal best-first search algorithms like Greedy Best-First Search are frequently used when building automated planning systems, their greedy nature can make them susceptible to being easily misled by flawed heuristics. This weakness has motivated the development of best-first search variants like ϵ-greedy node selection, type-based exploration, and diverse best-first search, which all use random exploration to mitigate the impact of heuristic error. In this paper, we provide a theoretical justification for this increased robustness by formally analyzing how these algorithms behave on infinite graphs. In particular, we show that when using these approaches on any infinite graph, the probability of not finding a solution can be made arbitrarily small given enough time. This result is shown to hold for a class of algorithms that includes the three mentioned above, regardless of how misleading the heuristic is.



(a) Problem graph for GBFS. (b) As a subgraph.

Figure 1: Graphs demonstrating the weaknesses of GBFS.

1 Introduction

Greedy best-first search (GBFS) (Doran and Michie 1966) is a popular suboptimal search algorithm that is often used in planning systems. GBFS takes a greedy approach to explore a state-space as it always prioritizes the node with the lowest heuristic value without any consideration of the cost to get to that node. While this greedy approach can be effective in practice, it can also make GBFS susceptible to being misled by an arbitrary amount if the heuristic is wrong. For example, consider the task of finding a path from node v to node g in the graph shown in Figure 1a. In this graph, all nodes in the left subtree below v down to some depth d have a heuristic value of 4. Because GBFS considers only the heuristic when evaluating nodes, the algorithm must exhaustively search the entire subtree (or **heuristic plateau**) before expanding n and finding g. The more misleading the heuristic (*ie.* the larger the plateau), the longer it will take to solve this problem. For example, if $d = 10$, GBFS will expand a total of $1,024$ nodes before n, while it will need to expand a million nodes in the plateau if $d = 20$.

However, heuristics are not typically wrong everywhere in a given state-space, and the greediness of GBFS often allows the algorithm to quickly progress through large areas of the problem in which the heuristic generally guides the search in the right direction. For example, consider the more representative graph shown in Figure 1b in which the initial node is u and the graph in Figure 1a appears as a subproblem. In this new graph, the heuristic provides imperfect, but useful guidance along the path from u to v. As such, the greediness of GBFS will get the search near the goal node much more quickly than a less greedy algorithm such as A* or breadth-first search. It is only when the search reaches v that the progress of GBFS will "stall" because it must exhaustively search the plateau under v.

Several recent approaches have augmented GBFS with **random exploration** with the aim of maintaining the benefits of greediness, while also making the algorithm more robust to heuristic error. These techniques work by encouraging the search to occasionally ignore the advice of the heuristic. For example, consider **ϵ-greedy node selection** (Valenzano et al. 2014). On every iteration of a GBFS that uses this technique, the algorithm will expand the node with the lowest heuristic value with probability $(1 - \epsilon)$, where ϵ is a parameter such that $0 \leq \epsilon \leq 1$. Otherwise, a node is selected for expansion uniformly at random from amongst those in the open list. Because the resulting algorithm is still greedy for a fraction $(1 - \epsilon)$ of the time, the search will still benefit from the value of greediness and quickly pass through areas of the state-space — like from u to v in Figure 1b – in which the heuristic provides useful guidance. How-

ever, by ignoring the heuristic with probability ϵ, the algorithm will not be as easily misled by heuristic error. To see this, consider this algorithm's performance on the problem in Figure 1a. In particular, let us compute the probability that n has not been expanded after some given time assuming the worst case for this graph: the plateau is infinitely big[1]. If $\epsilon = 0.5$, the probability that the algorithm will not have expanded n after 1024 iterations is 0.04, which decreases to 0.001 after a million expansions.

If the plateau is not infinite but only goes to a depth of d, or not all nodes in the plateau have a heuristic value of 4 or less, then the probability that n is not expanded in at most 1024 expansions is clearly still at most 0.04. It is therefore likely to have solved the problem in at most a thousand expansions, regardless of how misleading the heuristic is. In contrast, standard GBFS will only solve this problem in that amount of time if the heuristic is not too flawed.

In this paper, we formally show that the form of robustness seen above in the graph in Figure 1a also holds in general for ϵ-greedy node selection, and two other exploration-based algorithms: **type-based exploration** (Xie et al. 2014) and **diverse best-first search** (Imai and Kishimoto 2011). In particular, we show that these algorithms are **probabilistically complete** on any infinite graph-search problem. This means that the probability that a solution has been found can be made arbitrarily close to 1 given enough time. The paper therefore provides a theoretical justification for the claim that the exploration-based techniques are better able to handle heuristic error than standard GBFS.

2 Related Work

Several previous papers have investigated the completeness of search algorithms on infinite graphs, including the original paper on A* (Hart, Nilsson, and Raphael 1968). Dechter and Pearl (1985) later generalized this result by identifying sufficient conditions for guaranteeing the completeness of a best-first search on an infinite graph. This result is reviewed in Section 3.4. However, we note that GBFS does not satisfy these conditions, which is why it is incomplete on infinite graphs like that shown in Figure 1a.

The notion of probabilistic completeness has previously been considered in several fields. Hoos (1999) classified several stochastic local search algorithms based on whether they are probabilistically complete on SAT problems. LaValle and Kuffner (2001) have demonstrated that the RRT algorithm is probabilistically complete on kinodynamic motion planning problems. However, there are no previous results on the probabilistic completeness of the exploration-based best-first search variants.

Previous studies on exploration-based best-first search variants have empirically demonstrated that these techniques can greatly improve the performance of automated planners (Imai and Kishimoto 2011; Valenzano et al. 2014; Xie et al. 2014). These studies generally suggest that this improvement occurs because the exploration-based techniques more capably handle heuristic error. In this paper, we use infinite graphs to provide theoretical evidence of this claim.

Several additional techniques have introduced **diversity** into search through local searches (Xie, Müller, and Holte 2014; 2015) or by simultaneously expanding multiple nodes (Felner, Kraus, and Korf 2003). While a complete discussion of these approaches is beyond the scope of this paper, we note that these approaches are neither complete nor probabilistically complete on infinite graphs similar to those in Figure 1. These techniques may therefore benefit from additional random exploration as a way to make them more robust in the face of misleading heuristics. However, such analysis is left as future work.

Finally, Arvand is a random-walk based approach that can also help overcome heuristic error (Nakhost and Müller 2009). As it uses a very different form of search than the algorithms considered in this paper, we leave analysis of when this algorithm is probabilistically complete as future work.

3 Background and Terminology

In this section, we define the necessary notation and terminology used throughout the paper and describe an algorithm framework that we use in the analysis below.

3.1 Graph-Search Terminology

The focus of this paper is on problems that can be represented as pathfinding in a graph $G(V, E)$, where V is the set of **nodes** of G and E is the set of **edges**. All edges are assumed to be directed and to have a non-negative cost. If there is an edge $(n_p, n_c) \in E$ from n_p to n_c, the cost of the edge is denoted by $\kappa(n_p, n_c)$, n_c is called a **child** of n_p, and n_p is the **parent** of n_c. The set of children for any $n \in V$ is denoted by $\mathrm{children}(n)$. A **path** is a sequence of nodes $[n_0, ..., n_k]$ where $n_{i+1} \in \mathrm{children}(n_i)$ for any $0 \leq i < k$ and $n_i \neq n_j$ for any $0 \leq i < j \leq k$. The cost of a path P is given by the sum of the costs of the edges along P.

We now define a **graph-search problem** as the tuple $\tau = (G, n_{\mathrm{init}}, V_{\mathrm{goals}})$, where $n_{\mathrm{init}} \in V$ is the **initial node** and $V_{\mathrm{goals}} \subseteq V$ are the **goal nodes**. The objective of τ is to find a **solution path** $P = [n_0, ..., n_k]$ where $n_0 = n_{\mathrm{init}}$ and $n_k \in V_{\mathrm{goals}}$. τ is said to be **solvable** if such a path exists.

In this paper, we assume that $\mathrm{children}(n)$ is finite and bounded for any $n \in V$, even if V is infinite. This means that for any $n \in V$, $|children(n)| \leq B_M$ for some graph-specific positive constant B_M. B_M will be called the **maximum branching factor** of G.

3.2 Open-Closed List (OCL) Algorithms

We now introduce the **Open-Closed List (OCL) framework**. This framework is a generalization of A* (Hart, Nilsson, and Raphael 1968) that defines a class of algorithms that also includes WA* (Pohl 1970), GBFS, EES (Thayer and Ruml 2011), and the existing exploration-based best-first search variants that are the focus of this paper. Versions of these algorithms enhanced with multi-heuristic best-first search and preferred operator lists (Helmert 2006; Röger and Helmert 2010), are also in this class.

[1] We use the expansion time of n to measure performance since it is easier to calculate, and is clearly the node preventing progress.

Algorithm 1 The OCL Algorithm Framework

```
1: g(n_init) = 0, parent(n_init) = NONE
2: OPEN ← {n_init}, CLOSED ← {}
3: while OPEN is not empty do
4:    n ← SelectNode(OPEN)
5:    if n is a goal node then
6:       return solution path extracted from CLOSED
7:    for all n_c ∈ children(n) do
8:       if n_c ∈ OPEN then
9:          if g(n) + κ(n, n_c) < g(n_c) then
10:            g(n_c) = g(n) + κ(n, n_c), parent(n_c) ← n
11:      else if n_c ∈ CLOSED then
12:         if g(n) + κ(n, n_c) < g(n_c) then
13:            g(n_c) = g(n) + κ(n, n_c), parent(n_c) ← n
14:            CLOSED ← CLOSED − {n_c}
15:            OPEN ← OPEN ∪ {n_c}
16:      else
17:         g(n_c) = g(n) + κ(n, n_c), parent(n_c) ← n
18:         OPEN ← OPEN ∪ {n_c}
19:   CLOSED ← CLOSED ∪ {n}
20: return no solution exists
```

Pseudocode for the OCL framework is given in Algorithm 1. Like A*, all OCL algorithms search for a solution by iteratively building up a set of candidate paths through the expansion of nodes. These paths are maintained using OPEN and CLOSED lists, as well as g-costs and parent pointers just as A* does. The only difference between the pseudocode given for the OCL framework and a standard A* definition is that the OCL framework allows for the use of any arbitrary **policy** for iteratively selecting the next node for expansion. This policy is referred to as the SelectNode function in line 4. For example, the SelectNode function used by A* is defined to return the state in OPEN with the minimum value of $g(n) + h(n)$, where h is the heuristic function. However, the OCL framework allows for many other policies as well, including a GBFS that uses ϵ-greedy node selection. In that OCL algorithm, SelectNode is a stochastic function that returns the node in OPEN with the minimum heuristic value with probability $(1 - \epsilon)$, and an arbitrarily chosen node in OPEN with probability ϵ.

Best-first search is a common type of OCL algorithm that uses an evaluation function $F(.)$ to define the node selection policy. On every iteration, the node in OPEN with the minimal F-value is selected for expansion. For example, A* is a best-first search with evaluation function $F(n) = g(n) + h(n)$, while GBFS uses $F(n) = h(n)$.

In Algorithm 1, the OCL framework is defined so that nodes are always moved back from CLOSED to OPEN when a lower cost path is found to them (see lines 11 to 15). However, these **re-openings** are not necessary for the analysis below, as the proofs also hold if a node in CLOSED is never re-opened or even occasionally re-opened.

Finally, we say that an OCL algorithm is on iteration t if it has already expanded t nodes, and we denote the OPEN and CLOSED lists after t expansions as OPEN$_t$ and CLOSED$_t$, respectively. Note, we omit the subscript t, when the iteration in question is clear from the context.

3.3 Algorithm Completeness

Let us now formally define two types of termination conditions for OCL algorithms. The first is the standard notion of **algorithm completeness**, defined as follows:

Definition 1. *An OCL algorithm A is **complete** for a solvable graph-search problem τ if there exists an integer $t_{\max} \geq 1$ such that A is guaranteed to return a solution for τ in at most $t_{\max}$ iterations.*

On an infinite graph, algorithms like GBFS with ϵ-greedy node selection will rarely satisfy this definition since there is always a chance that they will be "unlucky". For example, on any problem with an infinitely long path P^∞, there is always a non-zero probability that for any t, the first t node expansions will all be nodes from P^∞. However, it can be shown that the likelihood that such exploration-based algorithms miss a solution due to being unlucky can be made arbitrarily small given enough time. This second type of termination guarantee is formally defined as follows[2]:

Definition 2. *An OCL algorithm A is **probabilistically complete** on a solvable graph-search problem τ if the probability that A solves τ in at most $t \geq 0$ node expansions converges to 1 as $t \to \infty$.*

3.4 Complete OCL Algorithms

Different OCL algorithms have previously been shown to be complete in certain specific cases. For example, A* is known to be complete on any finite graph, since in such graphs, there are only a finite number of paths from n_{init} to any other node in the state-space (Hart, Nilsson, and Raphael 1968). As such, even if A* exhaustively searches every path from n_{init} to every other node before expanding a goal node, the algorithm will still only expand a finite number of nodes before finding a solution. Since this argument also applies for any node selection policy, it is also true that any OCL algorithm is complete on any finite graph.

In infinite graphs, the only known completeness results for OCL algorithms are those given by Dechter and Pearl (1985) for best-first search algorithms. In particular, they showed that best-first search algorithms are complete in any graph in which there is no infinite path with a bounded $F(.)$ evaluation. This condition guarantees completeness by ensuring that the algorithm never gets stuck only considering the nodes along some infinite path. For example, consider the graph in Figure 1a and assume the plateau is infinite and all edges have a cost of 1. When using A* on this problem, the evaluation of n will be 6. As such, the only nodes in the plateau that may be expanded before n are those with an evaluation of at most 6. This rules out all nodes in the plateau that are a depth of 3 or more below v.

In contrast, the GBFS evaluation function is bounded along any infinite path in the plateau in Figure 1a, since the evaluation is never larger than 4. As such, GBFS will get stuck exploring down an infinitely long path in this graph, and is therefore incomplete. Moreover, notice that this issue

[2]This definition is a simplification of the notion of **probabilistically approximate completeness** given by Hoos (1999), which differentiated between optimal and suboptimal solutions.

that GBFS has with infinite graphs corresponds exactly with the issue it has with arbitrarily misleading heuristics.

4 Probabilistically Complete OCL Algorithms

In this section, we identify a sufficient condition that guarantees that an OCL algorithm is probabilistically complete on any graph. This condition is introduced in Section 4.1. We then show that algorithms satisfying this condition are probabilistically complete in Section 4.2. A simple extension of this result is then identified in Section 4.3.

4.1 Fair OCL Algorithms

We begin by introducing a property for OCL algorithms that guarantees probabilistic completeness on any graph. Intuitively, this property ensures that the probability of expanding a given node in OPEN is never disproportionately low. For example, consider a GBFS enhanced with ϵ-greedy node selection. Suppose that a node n is in OPEN after t expansions. The probability that n is the $t+1$-st node expanded is at least as large as $\epsilon/|\text{OPEN}_t|$, with equality if n does not have the smallest heuristic value among all nodes in OPEN_t. Since this is true for any t' when $n \in OPEN_{t'}$, the probability that n is expanded next is therefore always at least inversely proportional to the size of OPEN.

Let us now formalize this property as follows:

Definition 3. *An OCL algorithm is ϕ-**fair** to a node n where $0 \leq \phi \leq 1$ if it is true that if $n \in \text{OPEN}_t$ for any t, the probability that n is the $t+1$-st node expanded is at least*

$$\frac{\phi}{|\text{OPEN}_t|}$$

We refer to ϕ as the **fairness factor** of A for n, and say that A is fair to n if there exists a constant $0 < \phi \leq 0$ such that A is ϕ-fair to n. For example, a GBFS enhanced with ϵ-greedy node selection is fair to any node n if $\epsilon > 0$, since it is ϵ-fair to n with the positive fairness factor of ϵ. Notice that an algorithm is only said to be fair to a node if it is ϕ-fair to that node for some positive ϕ. This is because 0-fairness does not imply that the probability that the node is selected for expansion is always at least inversely proportional to the size of OPEN. We also refer to an OCL algorithm as being fair to a set $V' \subseteq V$ of nodes if there exists a fairness factor ϕ where $0 < \phi \leq 1$ such that A is ϕ-fair to all nodes in V'.

4.2 Theoretical Analysis of Fair OCL Algorithms

In this section, we prove that if an OCL algorithm is fair to all the nodes along some solution path of a given problem τ, then that algorithm is probabilistically complete on τ. The proof consists of two main steps. First, we show that if an OCL algorithm generates a node n that it is fair to, then the probability that n is eventually expanded converges to 1. This result is then inductively used to show the desired result: if the algorithm is fair to all of the nodes along some solution path, then the probability that a solution is eventually found also converges to 1.

Recall that when using ϵ-greedy node selection on the graph in Figure 1a, the probability that n is expanded converges to 1. The first lemma formally guarantees that this convergence happens in general for any OCL algorithm that is fair to any n, regardless of the size of OPEN.[3] The general structure of the proof is as follows. First, we find an upper bound on the probability that a fair OCL algorithm has not expanded a node n in the e iterations after it is generated. This upper bound is given as the product of the probabilities that it does not expand n on each of the e iterations. We then use the following well-known result from real analysis to show this product converges to 0 as $e \to \infty$.

Lemma 4.1. *For a given infinite sequence $a_0, a_1, ...$, of real numbers, the product $\prod_{j=0}^{\infty}(1+a_j)$ converges to a non-zero value if and only if $\sum_{j=0}^{\infty}|a_j|$ converges to a real number.*

Showing that the upper bound on the probability that n is not expanded converges to 0 then clearly implies that the probability n is expanded eventually converges to 1. This argument is formalized as follows:

Lemma 4.2. *Let n be a node in a given problem τ, and let A be an OCL algorithm that is fair to n. If n is in OPEN_t, then the probability that either n or a goal node is expanded in at most $t+e$ iterations will converge to 1 as $e \to \infty$.*

Proof. Assume n is in OPEN_t of A, and A is ϕ-fair to n for some constant $\phi > 0$. Let X_n be the random variable for the number of additional iterations it takes to expand either n or a goal node for the first time. This means that $\Pr(X_n > e)$ is the probability that neither n nor a goal node has been expanded after a total of $t+e$ expansions. By using the chain rule and the definition of $\Pr(X_n > e')$ for any e', we can perform the following derivation on $\Pr(X_n > e)$:

$$\begin{aligned}
\Pr(X_n > e) &= \Pr(X_n \neq 1 \wedge ... \wedge X_n \neq e) && (1)\\
&= \Pr(X_n \neq 1) \cdot (X \neq 2 | X_n \neq 1) \cdot ... \cdot \\
&\qquad \Pr(X \neq e | X_n \neq 1 \wedge ... \wedge X_n \neq e-1) && (2)\\
&= \Pr(X_n \neq 1 | X_n > 0) \cdot (X \neq 2 | X_n > 1) \cdot ... \cdot \\
&\qquad \Pr(X \neq e | X_n > e-1) && (3)\\
&= \prod_{i=1}^{e} \Pr(X_n \neq i | X_n > i-1) && (4)\\
&= \prod_{i=1}^{e} (1 - \Pr(X_n = i | X_n > i-1)) && (5)
\end{aligned}$$

By Line 5, an upper bound for $\Pr(X_n > e)$ can be given using lower bounds on $\Pr(X_n = i | X_n > i-1)$ for each i. Notice that if neither n nor a goal node has been expanded after $t+i-1$ expansions, then the probability of n being the $t+i$-th node expanded is at least $\phi / |OPEN_{t+e-1}|$ since A is ϕ-fair to n. While $|OPEN_{t+e-1}|$ is unknown, the following is an upper bound:

$$|\text{OPEN}_t| + (e-1) \cdot (B_M - 1) \qquad (6)$$

[3] For technical reasons, the lemma actually guarantees that the likelihood that either n or a goal node is expanded converges to 1.

This holds since each of the $(e-1)$ expansions removes one node from OPEN and adds at most B_M. By substitution into line 5, we are then left with the following inequality:

$$\Pr(X_n > e) \leq \prod_{i=1}^{e} \left(1 - \frac{\phi}{|\text{OPEN}_t| + (i-1)\cdot(B_M - 1)}\right) \tag{7}$$

We can therefore show that $\Pr(X_n > e)$ converges to 0 as $e \to \infty$ by showing that the right side of this inequality converges to 0. To do so, we consider the following summation so that we can later apply Lemma 4.1. Note, the derivation that follows consists of only algebraic manipulations.

$$\sum_{i=1}^{\infty} \frac{\phi}{|\text{OPEN}_t| + (i-1)\cdot(B_M - 1)} \tag{8}$$

$$= \frac{\phi}{B_M - 1} \cdot \sum_{i=1}^{\infty} \frac{1}{|\text{OPEN}_t|/(B_M - 1) + i - 1} \tag{9}$$

$$\geq \frac{\phi}{B_M - 1} \cdot \sum_{i=1}^{\infty} \frac{1}{\lceil |\text{OPEN}_t|/(B_M - 1) \rceil + i - 1} \tag{10}$$

$$\geq \frac{\phi}{B_M - 1} \cdot \sum_{i=\lceil |\text{OPEN}_t|/(B_M-1) \rceil}^{\infty} \frac{1}{i} \to \infty \tag{11}$$

Line 11 diverges since the harmonic series diverges, so the expression in line 8 must also diverge. By Lemma 4.1, this implies that the right side of the inequality in line 7 must either diverge to ∞ or converge to 0 as $e \to \infty$. Since all of the terms in this product are in the range $[0, 1]$, the product in line 4.1 cannot diverge to ∞, and it must converge to 0. Therefore $\Pr(X_n > e)$ converges to 0 as $j \to \infty$ by line 7, and so $\Pr(X_n \leq e)$ converges to 1. □

Notice that in the above proof, after every iteration during which n is not expanded, the probability that it is expanded on the next iteration can decrease if OPEN increases in size (see the paragraph following line 5). However, as the proof shows, ϕ-fairness is enough to ensure that this probability cannot decrease "too quickly", and so the probability that n is eventually expanded still converges to 1.

Let us now show that OCL algorithms that are fair to all the nodes along some solution path are probabilistically complete. Intuitively, the proof is based on the idea that Lemma 4.2 can be used inductively to show that over time, fair OCL algorithms will most likely continue to make progress along solution paths until a goal is found. For example, consider a problem with solution path $P = [n_0, ..., n_k]$. Lemma 4.2 states that once n_{k-1} is generated, it will probably be expanded eventually, thus generating n_k. Since the probability that n_k is expanded once it is generated converges to 1 regardless of when n_k is generated, together these facts imply that if n_{k-1} is generated, the likelihood that n_k is expanded converges to 1. The same can be shown to be true inductively for n_{k-2}, n_{k-3}, ..., n_0. This argument is formalized in the following lemma.

Lemma 4.3. *Let $P = [n_0, ..., n_k]$ be a solution path to problem τ, and let A be an OCL algorithm that is ϕ-fair to all nodes along P. If $n_j \in \text{OPEN}_t$ after t expansions for some $0 \leq j \leq k$, then the probability that A will expand a goal node in at most $t + e$ iterations converges to 1 as $e \to \infty$.*

Proof. The proof of this statement is by induction on j, starting with $j = k$ in the base case and considering decreasing values of j from there. Consider the base case where $j = k$. By Lemma 4.2, the probability that n_k or a goal node is expanded converges to 1 as the number of expansions converges to ∞. Since n_k is a goal node, this implies that the probability that a goal node will be expanded converges to 1. Therefore, the statement is true in the base case.

Now assume that the statement is true for all nodes on P at depth $j + 1$ or greater where $j \leq k$, and suppose that $n_j \in \text{OPEN}_t$ for some t. There are now two cases to consider.

In the first case, n_j is not the deepest node from P in OPEN after t expansions. As such, there is some $n_{j'}$ that is along P, such that $j' > j$ and $n_{j'}$ is in OPEN after t expansions. Since $j < j' \leq k$, the induction hypothesis implies that the probability that a goal node is expanded will converge to 1. Therefore, the statement is true in this case.

In the second case, n_j is the deepest node from P that is in OPEN_t. This implies that all nodes that are deeper than n_j along P have yet to be generated. We now introduce two random variables. First, let X_{goals} be the random variable for the number of additional iterations before a goal node is expanded. Second, let X_j be the random variable for the number of additional iterations before either n_j or a goal node is expanded. We will now show that $\Pr(X_{\text{goals}} \leq e)$ converges to 1 as $e \to \infty$ in this case. Since $X_j \leq X_{\text{goals}}$ and by the law of total probability, the following is equal to $\Pr(X_{\text{goals}} \leq e)$ as $e \to \infty$:

$$\lim_{e\to\infty} \sum_{i=1}^{e} \Pr(X_j = i) \cdot \Pr(X_{\text{goals}} \leq e | X_j = i) \tag{12}$$

Now consider $\Pr(X_{\text{goals}} \leq e | X_j = i)$. This is the probability that a goal node is expanded in at most e more expansions, given that n_j is the $t + i$-th node expanded. as $e \to \infty$. If n_j is the $t+i$-th node expanded, either n_{j+1} will be in OPEN_{t+i} (either because it is added to OPEN or was already there) or n_{j+1} will be in CLOSED_{t+i}. If $n_{j+1} \in \text{CLOSED}_{t+i}$, then it must have been expanded previously and so some deeper node from P must be in OPEN_{t+i}.[4] As such, regardless of whether n_{j+1} is in OPEN_{t+i} or CLOSED_{t+i}, there must be some $n_{j'}$ from P in OPEN_{t+i} where $j' > j$. Since the induction hypothesis guarantees that the probability that a goal node is expanded converges to 1 whenever a node deeper than n_j along P is in OPEN, this means that $\Pr(X_{\text{goals}} \leq e | X_j = i)$ converges to 1 as $e \to \infty$. Therefore, line 12 simplifies to the following:

$$\lim_{e\to\infty} \sum_{i=1}^{e} \Pr(X_j = i) \tag{13}$$

This summation is the probability that n_j or a goal node is eventually expanded, which converges to 1 by Lemma 4.2. Therefore $\Pr(X_{\text{goals}} \leq e)$ as $e \to \infty$ in this case.

[4] This is a well-known property of best-first search algorithms that also applies for all OCL algorithms.

Having handled all cases, the statement is true. □

Since the first node along any solution path is always in OPEN_0 (*ie.* before any nodes are expanded), Lemma 4.3 immediately implies the desired result:

Theorem 4.4. *An OCL algorithm A is probabilistically complete for any problem in which there is solution path P such that A is fair to all nodes along P.*

4.3 Periodically Fair OCL Algorithms

In this section, we show that Theorem 4.4 can be extended to algorithms that are not fair to the nodes along some solution path on every iteration, but are fair often enough. For example, consider the following variant of ϵ-greedy. Instead of always selecting a node uniformly at random from OPEN with probability ϵ, this variant always selects randomly on every third iteration and always selects according to the heuristic function on all other iterations. Clearly this new approach will behave similarly in practice to ϵ-greedy with $\epsilon = 1/3$, and so it is not surprising that this new algorithm is also probabilistically complete on any graph.

Unfortunately, Theorem 4.4 cannot be immediately used to imply this result, since this new variant is not fair to any node in OPEN on two out of every three expansions. However, the probability that any node $n \in \text{OPEN}_t$ will be selected for expansion at some point during the next three iterations is at least $1/[|\text{OPEN}_t| + 2 \cdot (B_M - 1)]$. This holds because a random selection must be made at some point during the next 3 expansions, and at that time there may be at most $2 \cdot (B_M - 1)$ new nodes in OPEN. As such, this ϵ-greedy variant will still select n sometime during the next 3 expansions with a probability that is at least inversely proportional to OPEN_t. Because this condition holds for any t, we say that this algorithm is **fair to n with a period** 3.

In general, if for some given problem τ there is a constant $\rho \geq 1$ such that OCL algorithm A is fair with a period of ρ to all the nodes on some solution path, then A is probabilistically complete on τ. While we omit the details, this proof requires only a minor modification to the proof above.

5 The Completeness of Existing Variants

In this section, we show that the existing best-first search variants that use exploration are probabilistically complete.

ϵ-greedy Node Selection. As shown in Section 4.1, GBFS enhanced with ϵ-greedy node selection is always fair to any node in OPEN. This is also true of any OCL algorithm that employs this technique. As such, any OCL algorithm enhanced with ϵ-greedy node selection is probabilistically complete on any graph.

Type-Based Exploration. The second GBFS variant we consider is type-based exploration (Xie et al. 2014). This algorithm selects greedily according to the heuristic on every second iteration, and probabilistically from OPEN on the remaining iterations. Unlike ϵ-greedy which selects uniformly at random from OPEN when making a probabilistic selection, the probabilistic selections in type-based exploration are made according to a distribution over OPEN that is defined by a **type system**. A type system is a function that partitions OPEN into disjoint and non-empty groups $T_0, ..., T_k$. Given a partitioning of OPEN, node selection is a two-step process. First, one of the partitions T_i is selected uniformly at random from among the k partitions. Secondly, a node is uniformly selected from among those nodes in T_i.

Let us now identify type system conditions that guarantee periodic fairness. To do so, notice that if OPEN is partitioned into ψ types, the probability that any n in OPEN is selected for expansion when a probabilistic expansion is made is $1/(\psi \cdot |T_i|)$, where T_i is the set of nodes of the same type as n. If there are at most K possible types for some finite K, then the probability that n is expanded is at least $1/(K \cdot |\text{OPEN}|)$. This holds since $\psi \leq K$ and T_i can contain at most all of the nodes in OPEN. In that case, the algorithm will be fair to any node. Type-based exploration will also be fair to any n on some solution path P if there is a finite upper bound Υ on the number of nodes that can ever be of the same type as n. In this case, the probability of selecting n when it is in OPEN will be at most $1/(|\text{OPEN}| \cdot \Upsilon)$, since $|T_i| \leq \Upsilon$ and the number of different types that OPEN will be partitioned into will be at most $|\text{OPEN}|$.

The theory presented above can therefore be used to guarantee that any OCL algorithm enhanced with type-based exploration will be probabilistically complete on any graph if the type system used has a maximum number of types or a maximum number of nodes of any type. For example, if the type system groups together all nodes with the same heuristic value and there are a finite number of heuristic values, the algorithm is periodically fair to any node and is therefore probabilistically complete. Alternatively, consider a type system that can only put two nodes into the same partition if they have the same depth from the initial node. When using this type system, there can be at most B_M^d nodes of the same type as any node at depth d of some solution path P. Since P has a finite length (*ie.* a maximum depth), the type-based search that uses this type system is periodically fair to any node on P. As such, the algorithm is probabilistically complete when using such a type system.

Diverse Best-First Search. The third best-first search variant considered is Diverse Best-First Search (DBFS) (Imai and Kishimoto 2011). While DBFS is an OCL algorithm, it is easier to understand DBFS as a sequence of **local searches** (GBFS specifically) that use a common CLOSED list. Each of these local searches begins with the selection of a node n from OPEN in a process explained below. Once n is selected, the rest of OPEN is temporarily ignored and a new GBFS is started with n as its initial node. This local GBFS prunes nodes previously seen in earlier local searches (*ie.* that are in the **global** CLOSED list), and executes until either a goal is found or some node expansion limit is reached. If the expansion limit is reached, the OPEN and CLOSED lists of the local GBFS are merged with the global OPEN and CLOSED lists collected from the previous local searches. A new node is then selected from OPEN to be the initial node of the next local search.

We now describe the policy used to select a node as the initial node for a local search. DBFS performs these selec-

tions stochastically using a type system that groups nodes together if they have the same depth from the global n_{init} and the same heuristic value. The type system is then used as follows. First, a type T_i is selected stochastically using a non-uniform distribution over the non-empty types. Then, a node is uniformly selected from amongst those of type T_i.

While we omit the details due to space constraints, the distribution used for type selection is designed to balance between favouring nodes at a shallower depth and favouring nodes with low heuristic values. Moreover, it can be shown that this distribution ensures that the algorithm is fair to any node that is along some solution path when selecting the initial node of a local GBFS. Intuitively, this holds because the distribution used is not too unbalanced over types, each of which can only contain a maximum number of nodes since the depth of a node is used to determine a node's type.

Since DBFS is fair on these selections, the algorithm can be shown to be periodically fair if these stochastic selections are done often enough. The frequency of these selections is determined by the expansion limits used during the local DBFS. In the original version of DBFS, this limit was set as the heuristic value of the node selected as the initial node of the local GBFS, where it is assumed that the heuristic values are all non-negative integers. If the heuristic value of any node in a given domain is at most M, then at most M expansions is made during any local search. A fair node selection is therefore made at least once every $M + 1$ expansions, and so DBFS is periodically fair in this case.

As such, DBFS is probabilistically complete on any graph if there is a maximum on the heuristic value of any node in the domain. Alternatively, DBFS is probabilistically complete on any graph if the node expansion limit is always set as some maximum value.

6 Conclusion

In this paper, we have considered the behaviour of several exploration-based best-first search algorithms on infinite graphs. In particular, we have shown that the probability that these algorithms will find a solution on infinite graphs becomes arbitrarily close to 1, given enough time. This result will hold regardless of the accuracy of the heuristic information used. Since GBFS is incomplete on these graphs for the same reason that it struggles with misleading heuristics, this result represents theoretical evidence regarding the increased robustness of the exploration-based techniques.

7 Acknowledgments

We thank the reviewers for their feedback on this paper. This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Dechter, R., and Pearl, J. 1985. Generalized Best-First Search Strategies and the Optimality of A*. *Journal of the ACM* 32(3):505–536.

Doran, J. E., and Michie, D. 1966. Experiments with the Graph Traverser Program. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 294(1437):235–259.

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.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2):100–107.

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

Hoos, H. H. 1999. On the Run-time Behaviour of Stochastic Local Search Algorithms for SAT. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, 661–666.

Imai, T., and Kishimoto, A. 2011. A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 985–991.

LaValle, S. M., and Jr., J. J. K. 2001. Randomized Kinodynamic Planning. *International Journal of Robotics Research* 20(5):378–400.

Nakhost, H., and Müller, M. 2009. Monte-Carlo Exploration for Deterministic Planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 1766–1771.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1(3-4):193–204.

Röger, G., and Helmert, M. 2010. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling*, 246–249.

Thayer, J. T., and Ruml, W. 2011. Bounded Suboptimal Search: A Direct Approach Using Inadmissible Estimates. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 674–679.

Valenzano, R.; Sturtevant, N. R.; Schaeffer, J.; and Xie, F. 2014. A Comparison of Knowledge-Based GBFS Enhancements and Knowledge-Free Exploration. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, 375–379.

Xie, F.; Müller, M.; Holte, R. C.; and Imai, T. 2014. Type-Based Exploration with Multiple Search Queues for Satisficing Planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2395–2402.

Xie, F.; Müller, M.; and Holte, R. C. 2014. Adding Local Exploration to Greedy Best-First Search in Satisficing Planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2388–2394.

Xie, F.; Müller, M.; and Holte, R. 2015. Understanding and Improving Local Exploration for GBFS. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling*, 244–248.