Breadth-First Search vs. Restarting Random Walks for Escaping Uninformed Heuristic Regions

Daniel Platnick[2, 1,*], Dawson Tomasz[1, 3], Eamon Earl[1, 3], Sourena Khanzadeh[1], Richard Valenzano[1, 3]

[1]Toronto Metropolitan University
[2]Flybits Labs
[3]Vector Institute
daniel.platnick@flybits.com, dawson.tomasz@proton.me, {eamon.earl, sourena.khanzadeh, rick.valenzano}@torontomu.ca

Abstract

Greedy search methods such as Greedy Best-First Search (GBFS) and Enforced Hill-Climbing (EHC) often struggle when faced with Uninformed Heuristic Regions (UHRs) like heuristic local minima or plateaus. In this work, we theoretically and empirically compare two popular methods for escaping UHRs: breadth-first search (BrFS) and restarting random walks (RRWs). First, we derive the expected runtime of escaping a UHR using BrFS and RRWs, based on properties of the UHR and the random walk procedure, and then use these results to identify when RRWs will be faster in expectation than BrFS. Next, we evaluate these methods for escaping UHRs by comparing standard EHC, which uses BrFS to escape UHRs, to variants of EHC called EHC-RRW, which use RRWs for that purpose. EHC-RRW is shown to have strong expected runtime guarantees in cases where EHC has previously been shown to be effective. Finally, we run experiments with these approaches on PDDL planning benchmarks to better understand their relative effectiveness for escaping UHRs.

1 Introduction

When given a reasonably accurate heuristic function, greedy algorithms like *Greedy Best First Search (GBFS)* (Doran and Michie 1966) and *Enforced Hill-Climbing (EHC)* (Hoffmann and Nebel 2001) can be effective at solving planning problems. However, when using a flawed heuristic function, these methods can become stalled due to *Uninformative Heuristic Regions (UHRs)* in which the heuristic provides no or flawed guidance. Notably, EHC is explicitly designed to perform a Breadth-First Search (BrFS) to find a way out of UHRs. When GBFS gets stuck in a *plateau* — which is a UHR in which all states have the same heuristic value — it too will degenerate into BrFS when using low-g tiebreaking.

Restarting Random Walks (RRW) have also been effectively used to escape UHRs. In GBFS, RRWs have been initiated when the search stops seeing improvement in the heuristic values of the states encountered (Xie, Müller, and Holte 2014). Alternatively, Arvand is an EHC-like local search that uses RRWs to progress through the state-space (Nakhost and Müller 2009, 2013).

In practice, BrFS-based approaches are more effective on some problems and RRW-based are more effective on others. However, more work is needed to improve our understanding of why these differences occur or when we would expect either method to be the better option. This work aims to help address this gap, with the goal of making it clearer when RRWs should be deployed, or when they should be used alongside BrFS in an *algorithm portfolio*. To that end, we make the following contributions:

1. We identify expected runtimes for BrFS and *constant-depth RRWs* for escaping a UHR with uniformly distributed exits. These results are given in terms of the size of the UHR and the *success probability*, which is the probability that a single random walk escapes the UHR.
2. Where the *goal depth* is the depth of the shallowest escape state, we show that RRWs will be faster in expectation than BrFS if the success probability is proportional to the ratio of the number of states at the goal depth to the number of states shallower than the goal depth. For unbiased random walks on directed trees, we also find a lower bound on the number of escape states required at the goal depth to guarantee RRWs are faster than BrFS.
3. We compare variants of EHC that use constant-depth RRWs or the popular Luby restart policy (Luby, Sinclair, and Zuckerman 1993) instead of BrFS to escape UHRs. We show that these methods have strong expected runtime guarantees in cases where EHC is known to be complete or have a polynomial runtime for STRIPS planning.
4. We empirically compare EHC with the RRW variants on PDDL problems to show how our theoretical results apply in practice, and how different state-space topological features relatively influence EHC and the RRW variants.

2 Preliminaries

In this section, we introduce our terminology and notation, and describe the algorithms and methods analyzed below.

2.1 Search Tasks and State-Space Topologies

A *state-space search task* $\mathcal{T}$ is defined by the tuple $\mathcal{T} = \langle \mathcal{S}, s_{\mathcal{I}}, \Delta, \Gamma \rangle$, where $\mathcal{S}$ is a finite set of *states*, $s_{\mathcal{I}}$ is the initial state, $\Delta : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ is the *state transition function*, and Γ :

*Work done while at Toronto Metropolitan University.

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Algorithm 1: Enforced Hill-Climbing

```
1: Input: task ⟨S, s_I, Δ, Γ⟩, heuristic h
2: P ← ⟨s_I⟩, s ← s_I
3: while True do
4:     T' ← ⟨S, s, Δ, Γ^h_s⟩
5:     P' ← brFS(T')
6:     if P' = ⟨⟩ then
7:         return ⟨⟩     % solution not found
8:     end if
9:     P ← P + P', s ← last(P')
10:    if Γ(last(P)) = True then
11:        return P
12:    end if
13: end while
14: return ⟨⟩     % No solution found
```

Algorithm 2: Restarting Random-Walks

```
1: Input: task ⟨S, s_I, Δ, Γ⟩
2: if Γ(s_I) = True then
3:     return ⟨s_I⟩     % Single state path is solution
4: end if
5: while True do
6:     P ← ⟨s_I⟩, s ← s_I, ℓ ← getDepth(), d ← 0
7:     while d < ℓ and |Δ(s)| > 0 do
8:         s' ← state sampled from Δ(s)
9:         P ← P + ⟨s'⟩
10:        if Γ(s') = True then
11:            return P
12:        end if
13:        s ← s', d ← d + 1
14:    end while
15: end while
```

$\mathcal{S} \to \{\text{True}, \text{False}\}$ is the *goal test function*. If s' is in $\Delta(s)$, we refer to s' as a *successor* of s. When Δ is used to find the successors of s, we say those successors are *generated*.

A path $P = \langle s_0, \ldots, s_k \rangle$ is a sequence of states where $s_i \in \Delta(s_{i-1})$ for every $0 < i \leq k$. The objective of a given task $\mathcal{T}$ is to find a *solution path*, where $s_0 = s_\mathcal{I}$ and s_k is a *goal state* (*ie.* $\Gamma(s_k) = \text{True}$). As our focus is on *satisficing* search — in which we want to find any solution regardless of cost — we ignore transition costs in this work. In addition, we let $last(P)$ denote the last state on P (*ie.* $last(P) = s_k$). If $P' = \langle s'_0, \ldots, s'_j \rangle$, then $P + P'$ is the concatenation of the paths P and P'. For brevity, we abuse notation and let $P + P' = \langle s_0, ..., s_{k-1}, s'_0, \ldots s'_j \rangle$ if the last state on P is the same as the first state on P' (*ie.* $s_k = s'_0$).

We now define several important state-space properties. For any $s \in \mathcal{S}$, the *depth* of s is the number of transitions in the shortest path from $s_\mathcal{I}$ to s. For example, $s_\mathcal{I}$ has a depth of 0, any state in $\Delta(s_\mathcal{I})$ has a depth of 1, and so on. A state s is called a *dead end* if no goal state is reachable from s. The *goal depth* d^* of a task $\mathcal{T}$ is defined as the minimum depth of any goal state. We also denote the set of unique states with a depth strictly less than d^* as $S_{<d^*} \subseteq \mathcal{S}$, and the set of unique states with a depth exactly equal to d^* as $S_{d^*} \subseteq \mathcal{S}$.

A *state-space topology* is a pair $\langle \mathcal{T}, h \rangle$, where $\mathcal{T}$ is a search task and $h : \mathcal{S} \to \mathbb{Z}^{\geq 0} \cup \{\infty\}$ is a *heuristic function*. Below, we assume that h never incorrectly identifies a state as a dead end, meaning if $h(s) = \infty$, then no goal state is reachable from s. While h will ideally provide useful search guidance, *Uninformative Heuristic Regions* (UHRs) do occur. A UHR around any state $s \in \mathcal{S}$ is the set of states reachable from s along any path P such that for any $s' \in P$, $h(s') \geq h(s)$. That is, no "heuristic progress" occurs while in a UHR. A state s_e in a UHR is called an *exit* if s_e has a successor s'' for which $h(s'') < h(s_e)$. We refer to any such "improving" successor s'' of s_e as an *escape state*. The length of the shortest path from s to any exit is also referred to as the *exit distance* of the UHR.

2.2 Search Algorithms and Methods

We now briefly describe the search methods of focus below.

Breadth-First Search (BrFS). We assume the reader's familiarity with BrFS, though pseudocode is given in Appendix A of the technical report associated with this work (Platnick et al. 2025). Notably, we define BrFS as a *best-first search* where a state's priority is given by its depth. BrFS is also defined to perform a goal test on a state s when it is generated, not when s is selected for expansion. BrFS will still find the shortest path when modified in this way.

Enforced Hill-Climbing (EHC). This local search method was originally used in the FF planner (Hoffmann and Nebel 2001). Given a state-space topology $\langle \mathcal{T}, h \rangle$, EHC performs a sequence of BrFSs, each aiming to escape the current UHR (see Algorithm 1). Importantly, instead of using the goal test Γ given for the overall task, each BrFS uses the following goal test function instead:

$$\Gamma^h_s(s') = \begin{cases} \text{True}, & \text{if } \Gamma(s') \text{ or } h(s') < h(s) \\ \text{False}, & \text{otherwise} \end{cases} \quad (1)$$

Γ^h_s succeeds when either a goal state according to Γ is found, or an escape state is found for the current UHR (*ie.* "heuristic progress" is made). Thus, EHC searches for a sequence of escape states until a goal state is reached.

Restarting Random Walks (RRWs). A *random walk* is a single path through a state-space that is generated stochastically (lines 5 to 12 of Algorithm 2). At every step of the walk, a successor of the last state is sampled and added to the current path. A random walk terminates when either a goal state is found, a state without any successors is encountered, or a maximum depth threshold is reached. A random walk is said to be *unbiased* if the states are sampled uniformly over the set of possible successors. We also let $0 \leq p_g \leq 1$ denote the *success probability* that the random walk will reach a goal. Note that p_g may depend on the structure of the state-space (*ie.* the distribution of goals) or the way successors are sampled for the random walk (*ie.* unbiased or biased).

A *restarting random walk (RRW)* performs a sequence of random walks, each starting from $s_\mathcal{I}$ (see Algorithm 2). An RRW terminates when any random walk reaches a goal state. The maximum length of each random walk is determined by a call to `getDepth()` (line 6). When using *constant-depth RRWs* — denoted as RRW^C_ℓ — `getDepth()` always returns the same integer constant $\ell > 0$.

The Luby Restart Policy. This strategy, which alters the depth limit from walk to walk, was originally defined for a general class of stochastic algorithms (Luby, Sinclair, and Zuckerman 1993). In the context of random walks, Luby, Sinclair, and Zuckerman showed that for any random walk procedure, there exists a constant $\ell^* > 0$, such that always restarting after ℓ^* steps has the minimum expected runtime over all possible *non-adaptive* restart policies. This means random walks are independent, and the restart policy does not change based on what is encountered during these walks.

Unfortunately, determining ℓ^* for a given task requires full knowledge of the runtime distribution of a single infinite length random walk on that problem. As this is not known prior to search, Luby, Sinclair, and Zuckerman introduced a general restart policy for unknown runtime distributions. We omit the full details of the policy, but note that it is based on a sequence whose first 15 values are $\langle 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \dots \rangle$. The resulting Luby restart policy is used for RRWs by having the length of the i-th random walk be the i-th value in the sequence. We refer to this algorithm as RRW$^{\mathcal{L}}$. Importantly, Luby, Sinclair, and Zuckerman showed that if T^* is the expected runtime when always restarting after ℓ^* steps, then the expected runtime when using the Luby sequence is $O(T^* \log T^*)$.

Intuitively, this approach performs longer and longer random walks to allow the search to reach deep goals if needed, while continually performing short walks to ensure shallow goals are not missed. In practice it is common to multiply all values in the sequence by some integer constant $m \geq 1$ to reach larger depths faster. This approach has the same runtime guarantees as when using the original sequence.

3 Expected Runtime Analysis

In this section, we characterize the expected runtime of BrFS and RRW$^C_\ell$ in terms of task size and random walk properties. We then find sufficient conditions that guarantee that RRW$^C_\ell$ will be faster in expectation than BrFS. This result is further refined in the case of unbiased random walks on a directed tree. We conclude the section with a discussion of the implications and limitations of this analysis.

We note that our results are given in terms of solving a search task, not just escaping a UHR. However, they also cover this case because the problem of escaping a UHR starting at state s can be modeled as a search task whose objective is to find a state with a lower heuristic value than $h(s)$ (*ie.* by using the goal test function in Equation 1).

Below, we use $B(\mathcal{T})$ and $R^C_\ell(\mathcal{T})$ for the random variables (RVs) of the runtime of BrFS and RRW$^C_\ell$, respectively. We measure runtime in terms of the number of goal tests performed or equivalently, the number of states generated.

3.1 General Runtime Analysis for BrFS

We begin with the following result for BrFS when the goal states are uniformly distributed at the goal depth:

Theorem 3.1. *If $\mathcal{T}$ has $g \geq 1$ goal states uniformly distributed among the $|S_{d^*}|$ states at the goal depth, then*

$$\mathbb{E}[B(\mathcal{T})] = |S_{<d^*}| + (|S_{d^*}| + 1)/(g + 1)$$

Proof. Let X be the number of goal tests that BrFS performs on states at depth d^*. Since BrFS examines all states shallower than d^* and none deeper than d^*, it follows that $B(\mathcal{T}) = |S_{<d^*}| + X$. Thus $\mathbb{E}[B(\mathcal{T})] = |S_{<d^*}| + \mathbb{E}[X]$ since $|S_{<d^*}|$ is a constant. Clearly $1 \leq X \leq |S_{d^*}|$ even if the goals are not uniformly distributed, since at least one state at the goal depth will be tested, and at worst all states at depth d^* will be tested. This means that

$$|S_{<d^*}| + 1 \leq B(\mathcal{T}) \leq |S_{<d^*}| + |S_{d^*}| \tag{2}$$

If the goal states are uniformly distributed at the goal depth, $\mathbb{E}[X]$ is equivalent to the expected number of selections needed when randomly picking states from the goal depth without replacement, until one of the g goal states is picked. Where s_i is any one of the $(|S_{d^*}| - g)$ non-goal states in S_{d^*}, let Z_i be an indicator RV for the event that s_i is picked before any of the g goals. Therefore, $\mathbb{E}[X] = \mathbb{E}[Z_1 + ... + Z_{|S_{d^*}|-g}] + 1$ since X is the number of non-goal states tested plus one for the selected goal state. Now notice that the next time that either one of the g goal states or s_i is generated, any of these $g + 1$ states is equally likely to be generated first. As such, $\mathbb{P}[Z_i] = 1/(g + 1)$, and the following holds:

$$\mathbb{E}[X] = 1 + \mathbb{E}[\sum_{i=1}^{|S_{d^*}|-g} Z_i] = 1 + \sum_{i=1}^{|S_{d^*}|-g} \mathbb{E}[Z_i] \tag{3}$$

$$= 1 + (|S_{d^*}| - g)/(g + 1) = (|S_{d^*}| + 1)/(g + 1) \tag{4}$$

Line 4 holds since the Z_i's are indicator variables and so $\mathbb{E}[Z_i] = \mathbb{P}[Z_i]$, and because we are summing over $(|S_{d^*}|-g)$ of RVs that all have the same expectation. Adding this to $|S_{<d^*}|$ yields the desired result. □

The theorem shows that the expected runtime decreases as the density of goals at depth d^* (*ie.* $g/|S_{d^*}|$) increases, since fewer states in S_{d^*} will likely need to be tested. However, regardless of this density, BrFS must still exhaustively examine all states shallower than d^* (see Equation 2).

3.2 General Runtime Analysis for RRWs

We now turn to characterizing the expected runtime of RRW$^C_\ell$ in the general case in terms of the success probability of any individual random walk. If the success probability of a single random walk is $p_g = 0$ (*ie.* $\ell < d^*$), then $\mathbb{E}[R^C_\ell(\mathcal{T})] = \infty$. Otherwise, we can say the following:

Theorem 3.2. *If the success probability of a random walk to depth ℓ on search task $\mathcal{T}$ is $p_g > 0$, then*

$$\mathbb{E}[R^C_\ell(\mathcal{T})] \leq \ell/p_g + 1$$

Proof. Let Y be the RV for the *number of random walks* it takes to find a goal when using RRW$^C_\ell$, L be the RV for the length of some random walk given that the walk reaches a goal, and $\bar{L}$ be the RV for the length of some random walk given it does not reach a goal. RRW$^C_\ell$ will perform $(Y - 1)$ walks of length $\bar{L}$ and one random walk of length L, and so

$$\mathbb{E}[R^C_\ell(\mathcal{T})] = \mathbb{E}[(Y - 1)\bar{L} + L + 1] \tag{5}$$

$$= \mathbb{E}[Y - 1]\mathbb{E}[\bar{L}] + \mathbb{E}[L] + 1 \tag{6}$$

$$= (1/p_g - 1)\mathbb{E}[\bar{L}] + \mathbb{E}[L] + 1 \tag{7}$$

$$\leq \ell/p_g + 1 \tag{8}$$

The additional 1 in line 5 comes from the single goal test of $s_{\mathcal{I}}$ on line 2 of Algorithm 2. The random walks themselves are independent and identically distributed (IID), and so the length of each walk that does not reach a goal is independent of the number performed. As such, Y and $\bar{L}$ are independent and Line 6 holds. The IID property also means that Y follows the geometric distribution, and so $\mathbb{E}[Y] = 1/p_g$ (line 7). The final line holds because all walks have a length of at most ℓ and so $\mathbb{E}[L]$ and $\mathbb{E}[\bar{L}]$ are both at most ℓ. □

Runtime on constant branching factor trees. To better understand Theorems 3.1 and 3.2, consider a search task $\mathcal{T}_b$ on a directed tree with constant branching factor $b \geq 2$, and $g \geq 1$ goals uniformly distributed at depth d^*. There are b^d states at every depth $d \geq 0$ of such a tree, meaning that $|S_{d^*}| = b^{d^*}$ and $|S_{<d^*}| = b^0 + b^1 + ... + b^{d^*-1} = (b^{d^*} - 1)/(b-1)$. If we are using unbiased random walks length $\ell \geq d^*$ on such a tree, then $p_g = g/b^{d^*}$. Therefore, Theorems 3.1, and 3.2 imply the following

$$\mathbb{E}[B(\mathcal{T}_b)] = (b^{d^*} - 1)/(b-1) + (b^{d^*} + 1)/(g+1) \quad (9)$$

$$\mathbb{E}[R^C_\ell(\mathcal{T}_b)] \leq \ell b^{d^*}/g + 1 \quad (10)$$

Figure 1a shows the expected runtime of BrFS and RRW^C_ℓ with unbiased walks on such a tree — as calculated using equations 9 and 10 — with a constant branching factor $b = 4$, $d^* = 6$, and different numbers of goals uniformly distributed among the 4096 states at depth 6. BrFS is significantly faster when there are very few goals, but RRW^C_ℓ quickly catches up as the number of goals increases, depending on ℓ. Intuitively, this is because BrFS must examine all states in $S_{<d^*}$ — shown as the dotted line — regardless of the goal density. RRW^C_ℓ has no such requirement, as its runtime converges to d^* (*ie.* the task is solved with the first random walk) as g increases to $|S_{d^*}|$. In the next section, we formalize a more general version of this relationship.

3.3 Comparative Runtime Analysis

We now use Theorems 3.1 and 3.2 to identify different sufficient conditions for which RRWs have a lower expected runtime than BrFS. For certain results, the full proofs can be found in a technical report associated with this work (Platnick et al. 2025).

We begin by finding a lower bound on the success probability p_g that leads to RRWs being faster than BrFS:

Theorem 3.3. *Let $\mathcal{T}$ be a search task. Then $\mathbb{E}[R^C_\ell(\mathcal{T})] \leq \mathbb{E}[B(\mathcal{T})]$ if for the success probability p_g of any single random walk, it holds that $p_g \geq \ell/|S_{<d^*}|$.*

Proof. Starting with the assumption on p_g above, the result follows from the derivation below.

$$p_g \geq \ell/|S_{<d^*}| \quad (11)$$

$$1/p_g \leq |S_{<d^*}|/\ell \quad (12)$$

$$\ell/p_g + 1 \leq |S_{<d^*}| + 1 \quad (13)$$

$$\mathbb{E}[R^C_\ell(\mathcal{T})] \leq \mathbb{E}[B(\mathcal{T}_s)] \quad (14)$$

Line 12 simple takes the inverse of the previous line. Line follows by multiplying both sides by ℓ and then adding 1 to both sides. The final line holds by Theorem 3.2, and by the fact that since BrFS must always expand at least one state at the goal depth, it holds that $\mathbb{E}[B(\mathcal{T}_s)] \geq |S_{<d^*}| + 1$. □

If we assume that the goal states are uniformly distributed at the goal depth, then we can refine this result as follows:

Theorem 3.4. *Let $\mathcal{T}$ be a search task with $g \geq 1$ goal states uniformly distributed among the $|S_{d^*}|$ states at goal depth $d^* \geq 1$. Then $\mathbb{E}[R^C_\ell(\mathcal{T})] \leq \mathbb{E}[B(\mathcal{T})]$ if the success probability p_g of any single random walk satisfies*

$$p_g \geq \frac{\ell}{|S_{<d^*}| + (|S_{d^*}| + 1)/(g+1) - 1}$$

The proof is almost identical to that given for Theorem 3.3 above, except it uses the bound for $\mathbb{E}[B(\mathcal{T}_s)]$ given by Theorem 3.4. The full proof appears in Appendix B of the technical report.

Next, we consider the case where the state-space has the structure of a directed tree and the random walks are unbiased. Notably, we do not assume a constant branching factor, and some states in the tree may not even have any successors. To handle this, let $p_{d^*} \geq 0$ be the probability of any single random walk reaching the goal depth. Here, $p_g \leq p_{d^*}$ and $p_g > 0$ since the walks are unbiased. In this scenario, we can provide the following lower bound on the number of goals at the goal depth needed to guarantee that RRW^C_ℓ is at least as fast in expectation as BrFS.

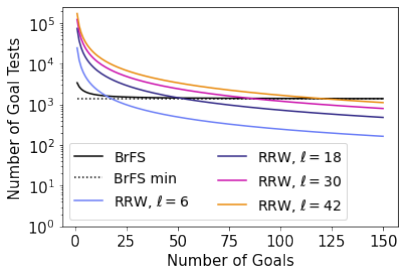
Theorem 3.5. *Let $\mathcal{T}$ be a search task on a tree with goal depth $d^* \geq 2$ and $1 \leq g < |S_{d^*}|$ goals at depth d^*. Let p_{d^*} be the probability of an unbiased random walk of length ℓ reaching the goal depth, where $0 < p_{d^*} \leq 1$. Then $\mathbb{E}[B(\mathcal{T}_b)] \geq \mathbb{E}[R^C_\ell(\mathcal{T}_b)]$ if the number of goals g at the goal depth satisfies $g \geq \ell|S_{d^*}|/\left[p_{d^*}|S_{<d^*}|\right]$, and those goals are uniformly distributed among all the states at the goal depth.*

Proof. In this case, $p_g \geq p_{d^*} g/|S_{d^*}|$, because the probability of reaching the goal depth is p_{d^*}, and the probability of visiting a goal state given that we have reached depth d^* is $g/|S_{d^*}|$. This is formally proven in Appendix B of the technical report. Thus, if the number of goals satisfies $g \geq \ell|S_{d^*}|/\left[p_{d^*}|S_{<d^*}|\right]$, then

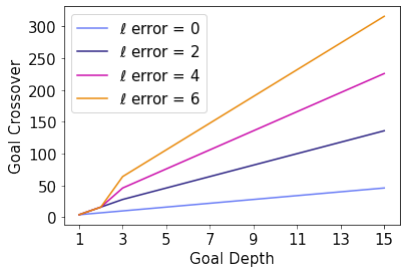
$$p_g \geq p_{d^*} g/|S_{d^*}| \geq p_{d^*} \frac{\ell|S_{d^*}|}{p_{d^*}|S_{<d^*}|}/|S_{d^*}| \geq \ell/|S_{<d^*}| \quad (15)$$

The result thus follows by Theorem 3.3. □

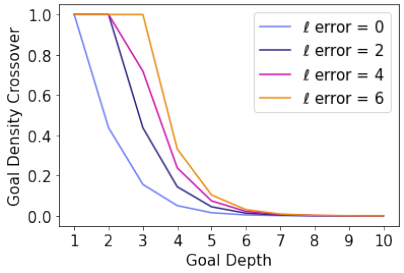
By appealing to Theorem 3.3 instead of Theorem 3.4, this proof is implicitly using 1 as a lower bound on the number of states BrFS sees at the goal depth. When there are uniformly distributed goals, this will underestimate the actual expected number of states seen at depth d^*, which is $(|S_{d^*}|+1)/(g+1)$ (Theorem 3.1). As a result, the bound on g given in Theorem 3.5 can overestimate the actual needed number of goals. A more accurate bound is given in Appendix B of the technical report, which uses a correction term to better account for the work that BrFS does at the goal depth. However, we focus on the simpler bound in Theorem 3.5 since the improvement in accuracy is marginal and



(a) Expected runtime for different numbers of goals where $d^* = 6$.



(b) The goal crossover point for different goal depths.



(c) The goal density crossover point for different goal depths.

Figure 1: BrFS and RRW_ℓ^C with unbiased random walks on a directed tree with a branching factor of 4.

this simpler bound already captures the main properties affecting the relative performance of BrFS and RRWs.

Two other notable cases are formalized in Appendix B of the technical report. First, we show that RRW_ℓ^C is usually faster than BrFS if all states in S_{d^*} are goals. This holds because the first random walk to reach the goal depth will succeed, while BrFS must still examine all states in $S_{<d^*}$. Next, we show that if the goal depth is 1, then BrFS is usually faster than RRW_ℓ^C using unbiased random walks. Intuitively, this is because BrFS examines the states in S_{d^*} one-by-one in turn, while RRW_ℓ^C will sample these states with replacement through the random walks.

3.4 Understanding the Theory

In this section, we consider the practical implications and the limitations of the runtime analysis given above.

The crossover point. Intuitively, Theorem 3.5 indicates that the number of goals needed for RRW_ℓ^C to outperform BrFS on a directed tree — which we call the *goal crossover point* — largely depends on the ratio of the number states at the goal depth, $|S_{d^*}|$, to the number of states shallower than the goal depth, $|S_{<d^*}|$. Implicitly, it also depends on d^* since $\ell \geq d^*$. For example, again consider a task on a directed tree with constant branching factor b and uniformly distributed goals. On such problems, $|S_{<d^*}| = (b^d - 1)/(b - 1)$ and $|S_{d^*}| = b^d$. It also holds that $p_{d^*} = 1.0$, since all states have at least one successor. Thus, Theorem 3.5 states that the goal crossover point is $\ell(b-1)b^d/(b^d-1)$. This is seen in Figure 1b, which shows the goal crossover point when $b = 4$, as a function of goal depth and "ℓ error". That is, we assume $\ell = (1.0 + e)d^*$, meaning each line corresponds to overestimating d^* by the same constant factor.

While the goal crossover point increases linearly with the goal depth, the density $g/|S_{d^*}|$ of goals at the goal depth needed for RRW_ℓ^C to outperform BrFS actually *decreases* with d^*. This is seen in Figure 1c, which shows the *goal density crossover* (*ie.* the goal crossover divided by $|S_{d^*}|$) on a directed tree with constant branching factor of 4.

Deeper goals and transpositions. While Theorem 3.5 captures the importance of the relationship between goal density and the ratio of $|S_{d^*}|/|S_{<d^*}|$, practical performance when escaping UHRs may differ for several reasons. First, RRW_ℓ^C can benefit from goals (*ie.* escape states) at depths between d^* and ℓ, as these goals will increase the success probability. BrFS does not benefit from such goals, since it never generates states deeper than depth d^*. However, BrFS will be much better at handling *transpositions*. Transpositions refers to the existence of multiple paths between different pairs of states. These do not significantly impact the performance of BrFS, since that algorithm uses duplicate detection to avoid ever expanding a state more than once.

In contrast, RRW_ℓ^C does not keep track of what parts of the state-space it has already visited, and so a single random walk may return to a state it has already visited. As such, when there are many transpositions, RRW_ℓ^C is effectively searching on a larger search tree than BrFS, meaning RRW_ℓ^C will likely struggle in a similar manner as IDA* (Korf 1985) does on such problems. Along with the above observations about the goal density crossover, we would therefore expect RRW_ℓ^C to more quickly escape UHRs in large combinatorial state-spaces, and BrFS to better handle cases with very few escape states or many transpositions.

Wall-clock time and memory. RRW_ℓ^C may also have an advantage in terms of wall-clock time since it does not have the overhead of maintaining open and closed lists as needed for duplicate checking and other operations. In PDDL planning, this overhead is likely minimal since runtime is dominated by heuristic calculation. However, the use of open and closed lists do mean the worst-case memory requirements of BrFS is $O(B^D)$, while it is only $O(\ell)$ for RRW_ℓ^C since only a single random walk is in memory at any one time. This can make RRW_ℓ^C especially useful in low-memory scenarios.

Uniformly distributed goals. Several of the results above rely on the assumption that the goals are uniformly distributed at the goal depth. In the case of BrFS, this assumption is used to calculate the expected number of states generated at the goal depth. When goals are not uniformly distributed, the accuracy of Theorems 3.1 and 3.4 will depend on how well $(|S_{d^*}| + 1)/(g + 1)$ estimates the number of goals seen at the goal depth. For example, Theorems 3.1 and 3.4 still hold if unbiased random tiebreaking is used to pick the order that states are expanded at within the same level, even without uniformly distributed goals. However, if some static tiebreaking is used that is well (or poorly) suited for

the given problem's goal distribution, these bounds would overestimate (underestimate) the expected runtime.

In the case of Theorem 3.5, the role of this assumption is that it is used to show the success probability p_g is at least as large as $\ell p_{d^*} g / |S_{d^*}|$. If the goal distribution is not uniform, the accuracy of this crossover point will therefore depend on how much p_g deviates from this value.

4 Enforced Hill-Climbing with RRWs

Recall that EHC breaks the search into a sequence of UHRs, where BrFS is used to find an escape state for each. Thus, we can study the relative effectiveness of BrFS and RRWs for escaping UHRs by comparing standard EHC to variants that use RRWs instead of BrFS to escape the UHRs. To that end, we introduce EHC-RRW$^C_\ell$ and EHC-RRW$^\mathcal{L}$, which only differ from EHC in that they call RRW$^C_\ell$ and RRW$^\mathcal{L}$, respectively, instead of BrFS on line 5 of Algorithm 1. In this section, we identify several formal properties for these variants, and evaluate them on PDDL problems.

4.1 Properties of EHC and EHC-RRW Variants

We now consider the EHC-RRW variants, and how their theoretical properties compare to the original EHC algorithm.

Relationship to Arvand. We first note that these EHC-RRW variants have a strong connection to Arvand (Nakhost and Müller 2009, 2013) which also performs an RRW-based local search. However, instead of committing to the first escape state found, Arvand commits to the state with the lowest heuristic value found after a fixed number of walks. Arvand also only calculates the heuristic value of the last state along every random walk, restarts the entire search back to $s_\mathcal{I}$ when progress stalls, and incorporates a number of other planning enhancements. While this makes Arvand a powerful planner, these features make it unsuitable for isolating and studying the effectiveness of using RRWs to escape random walks. Thus, this work focuses on the simpler EHC-RRW variants introduced above.

Termination guarantees. We will now consider problems on which EHC is known to be complete, and show that the EHC-RRW variants have a finite expected runtime on such problems. Our analysis is based on that of Hoffmann (2005) who outlined a domain taxonomy — originally for the *delete relaxation* h^+ heuristic — that categorizes domains according to their topological characteristics. The first main property considered in this taxonomy relates to dead-ends. A domain either has no dead-ends, *recognized* dead-ends (*ie.* s is a dead-end if and only if $h(s) = \infty$), or *unrecognized* dead-ends ($\exists s \in S$ such that s is a dead-end and $h(s) < \infty$). Hoffmann showed that EHC is complete on a state-space topology $\langle \mathcal{T}, h \rangle$ if $\mathcal{T}$ has no dead-end states, or all dead-end states are recognized by h. This is because a UHR in any such $\mathcal{T}$ must have a finite exit distance, where recognized dead-ends are treated as states with no successors. As such, there is some maximum exit distance $D(\mathcal{T})$ over all UHRs in $\mathcal{T}$. If B is the maximum number of successors of any state in $\mathcal{T}$, then $|S_{<d^*}| \in O(B^{D(\mathcal{T})})$ and $|S_{d^*}| \in O(B^{D(\mathcal{T})+1})$, where the extra "plus one" is because the shallowest escape is one step deeper than the shallowest exit. Thus, the runtime for each BrFS to escape a UHR will be $O(B^{D(\mathcal{T})+1})$ by Equation 2. Since h only returns non-negative integer values, there are at most $h(s_\mathcal{I})$ UHRs, and so the runtime of EHC is $O(h(s_\mathcal{I})B^{D(\mathcal{T})+1})$. Thus, EHC is complete on such state-space topologies, which we call *EHC-complete*. Domains with *unrecognized* dead ends are then *EHC-incomplete*.

Recall that an algorithm A is complete on a problem if there exists some constant $k \geq 0$, such that A always terminates on that problem in at most k steps. As the EHC-RRW variants are stochastic, no such constant exists for these methods. That said, the EHC-RRW variants using unbiased random walks can be shown to have a finite expected runtime on EHC-complete problems. To see why, consider using EHC-RRW$^C_\ell$ where $\ell \geq D(\mathcal{T}) + 1$. In this case, $p_g \geq 1/B^{D(\mathcal{T})+1}$ since at least one path with depth at most $D(\mathcal{T}) + 1$ will reach an escape. Thus, the expected runtime to escape any UHR will be $O(\ell B^{D(\mathcal{T})+1})$ by Theorem 3.2, and the expected runtime to solve the problem is $O(h(s_\mathcal{I})\ell B^{D(\mathcal{T})+1})$ by the same argument as EHC.

Since the expected runtime of using RRW$^\mathcal{L}$ to escape a UHR will be no worse than a log-factor more than the optimal restart policy on that UHR (Luby, Sinclair, and Zuckerman 1993), the expected runtime of EHC-RRW$^\mathcal{L}$ is at most a log-factor worse than EHC-RRW$^C_\ell$ with $\ell = D$, by a similar argument as above. As such, EHC-RRW$^C_\ell$ has a finite expected runtime if $\ell \geq D$ and EHC-RRW$^\mathcal{L}$ will always have a finite expected runtime on EHC-complete problems.

Bounded-UHR Problems. Next we consider problems on which EHC has polynomial runtime. Such problems were also identified in Hoffmann's taxonomy, which further divides the EHC-complete domains by UHR size. In *bounded-UHR* domains, there exists an integer $D > 0$ such that the exit distance of every UHR in every problem in the domain is at most D. In contrast, the exit distance of UHRs in *unbounded-UHR* domains can grow arbitrarily with problem size even for problems within the same domain.

To see the importance of bounded-UHRs, consider solving STRIPS planning problems — where the set of operators $\mathcal{O}$ is given as input — when using the h^+ heuristic. Here, $|\mathcal{O}| = B$, and since no operator in $\mathcal{O}$ can be included more than once in the optimal delete relaxed plan to a problem, $h^+(s_\mathcal{I}) \leq B$. Thus, because D is independent of the problem input, EHC has a polynomial runtime of $O(B \cdot B^{D+1})$ on problems from bounded-UHR domains.

By a similar analysis as above, EHC-RRW$^C_\ell$ will clearly have a polynomial expected runtime of $O(\ell B^{D+1})$ on such STRIPS planning problems if $\ell \geq D$, and EHC-RRW$^\mathcal{L}$ will always have a polynomial expected runtime in this case.

4.2 Empirical Evaluation

In this section, we evaluate EHC and EHC-RRW on PDDL planning problems to better understand the relative effectiveness of BrFS and RRWs for escaping UHRs. Where noted, additional details and results can be found in the technical report (Platnick et al. 2025) associated with this paper.

Autoscale Suite	Domain Classification	EHC	EHC-RRW$^C_\ell$ $\ell = 10$	$\ell = 25$	$\ell = 50$	EHC-RRW$^\mathcal{L}$ $m = 1$	$m = 2$	$m = 4$
Optimal Track	UHR-Bounded Total (180)	180.0	180.0	180.0	180.0	180.0	180.0	180.0
	UHR-Unbounded Total (240)	219.6	196.0	218.4	223.0	222.6	223.0	223.8
	Incomplete Total (90)	41.6	24.2	25.0	25.0	35.8	34.6	28.8
Agile Track	UHR-Bounded Total (153)	103.6	99.6	99.6	98.0	97.2	97.4	96.4
	UHR-Unbounded Total (220)	97.2	80.2	100.8	101.4	107.8	107.8	107.0
	Incomplete Total (90)	25.0	17.8	15.6	14.2	21.6	19.8	19.8

Table 1: A summary of the coverage of EHC and the EHC-RRW variants on different types of problems.

Experimental Setup. We tested all methods in Fast Downward (Helmert 2006). EHC was re-implemented since the existing version deviated from the original description in several important ways. Early experiments suggested our version outperforms the existing one. The details of our implementation and how it differed from the existing one can be found in Appendix C of the technical report.

The problems used for testing came from the optimal and agile Autoscale benchmark suites (Torralba, Seipp, and Sievers 2021). In particular, we used the 17 domains that have been previously categorized for h^+ according to the taxonomy described in Section 4.1 (Hoffmann 2005). The categorization of these 17 domains according to the h^+ taxonomy can be found in Appendix D of the technical report.

All algorithms were tested using the unit-cost FF heuristic (Hoffmann and Nebel 2001). While FF only approximates h^+, it has previously been shown empirically that the same taxonomy holds for FF on the 17 domains in question (Hoffmann 2001). In addition, if h^+ recognizes dead-ends in a domain, then provably so too will the FF heuristic, meaning completeness is not impacted by using FF (Hoffmann 2005).

Finally, all experiments were run on a VMware Virtual Platform using an 8-core Intel Xeon Gold 6226R CPU @ 2.90GHz with a 16 KB L1 cache, with a 10 minute time limit and a 3.5 GB memory limit per problem. Results were averaged over 5 runs per problem, including for EHC which was implemented to use random tie-breaking. EHC-RRW$^C_\ell$ and EHC-RRW$^\mathcal{L}$ were each tested with three different values for the walk length ℓ and the base multiplier m, respectively.

Coverage Results. Table 1 summarizes the coverage results of the different methods on the different test suites used. Full per-domain results can be seen in Appendix D of the technical report. The number of problems per category is shown in parentheses. The technical report also contains plots that show how coverage changes with the number of evaluations and time. While Autoscale contains 30 problems, Fast Downward was unable to translate some problems in the agile track from PDDL to SAS+ in the given 3.5 GB memory limit. We omit these from the test set.

Next, we consider each taxonomy category individually to better understand how different topological state-space features impact performance.

Bounded-UHR Domains. All optimal track problems in bounded-UHR domains were solved by all tested methods. This is consistent with the strong expected runtime guarantees we have for EHC and the EHC-RRW variants. However, the large size of the agile track problems meant they still remain challenging for EHC-based approaches. For example, the `logistics` domain — which has a maximum exit distance of 1 — has such a high branching factor that only tens of states were being expanded per second.

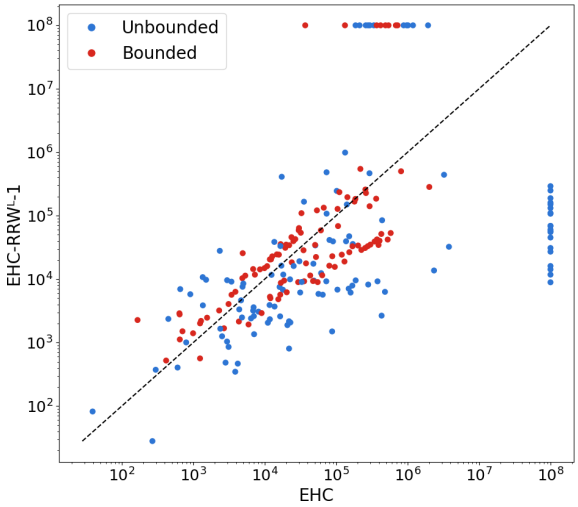


Figure 2: Per-problem runtime comparison between EHC and EHC-RRW$^\mathcal{L}$ on the EHC-complete agile problems.

A comparison of per-problem heuristic evaluations (*ie.* runtime) of one run of each of EHC and EHC-RRW$^\mathcal{L}$ with $m = 1$ on the complete Agile track domains is shown in Figure 2. The bounded-UHR problems are shown as red dots. For these runs, the two variants performed about equal in terms of runtime, with EHC having slightly better coverage. This is consistent with our theory which suggests a high goal density is needed for RRW$^C_\ell$ to be faster if the escapes are shallow (*ie.* see Figure 1c). The figure also shows there is significant correlation between the performance of these methods, which we verified by calculating the Pearson correlation of the logarithm of these evaluation counts. The correlation on problems solved by both methods was 0.81.

Unbounded-UHR Domains. EHC-RRW$^\mathcal{L}$ has the best coverage in these domains, but this is largely due to the two `pipesworld` domains (see full results in the technical report). As seen in Figure 2 (unbounded-UHR problems are shown in blue), EHC-RRW$^\mathcal{L}$ also showed improved runtime

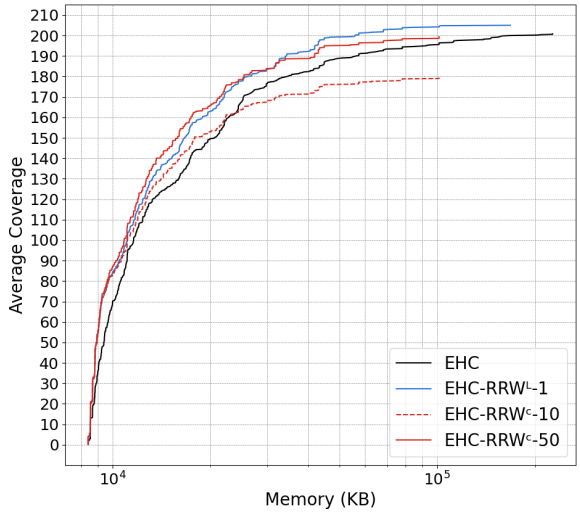


Figure 3: Memory usage comparison between EHC and the EHC-RRW variants on the EHC-complete agile problems.

and was faster or the only one able to solve 70.3% of the unbounded problems when compared to EHC on the agile track problems. Given our theoretical results, this suggests the goal density is not dropping dramatically as the exit distance increases. EHC-RRW$^{C}_{\ell}$ can show strong coverage, but is very sensitive to the selection of ℓ.

The Pearson correlation between EHC-RRW$^{\mathcal{L}}$ and EHC on unbounded-UHR problems was 0.62. While this is lower than on bounded-UHR problems, it is still reasonably high.

EHC-Incomplete Domains. Table 1 shows the EHC-based approaches struggle the most on domains with unrecognized dead ends. EHC has the best performance, albeit on only 3 domains. In such domains, it is not only important to escape UHRs as quickly as possible, but to find a "good" escape that does not lead to a dead end region of the state-space. We hypothesize that when using h^{+}-based heuristics on such problems, the use of BrFS to find shallowest escapes may have an advantage. This is because delete relaxation based methods do not recognize when resources (like fuel) are exhausted by an action, but shallower escapes may mean less resources are being used up, and thus EHC is less likely to find an escape leading to a dead end. However, further investigation is needed on this topic.

Memory Usage. Figure 3 shows how coverage relates to memory for the tested methods on the agile EHC-complete domains. For readability, we only include $m = 1$ for EHC-RRW$^{\mathcal{L}}$ since all values had very similar behaviour, and the best and worst performing values of ℓ for EHC-RRW$^{C}_{\ell}$. The EHC-RRW variants show clear advantages in this evaluation, which aligns with the fact that RRWs do not need to maintain open and closed lists like BrFS. Notably, none of the tested methods ran out of memory on any problem.

5 Related Work

Nakhost and Müller (2014) formally analyzed the expected runtime of a single random walk and RRWs on classes of undirected graphs characterized by the probability of the child of any vertex being closer or farther from a goal than its parent. Their model for RRWs assumed a constant restart probability at every step. In contrast, we assume the walks all restart at a constant restart depth, and our expected runtime results are given in terms of this restart depth, the goal depth, and either the probability that a single random walk reaches the goal, or the density of goals at the goal depth.

Everitt and Hutter (2015) theoretically compared BrFS to depth-first search on bounded depth-trees. Their analysis does not include RRWs, which are better suited for escaping UHRs than depth-first search. They also make different assumptions on the distribution of the goals in the tree. Understanding the applicability of their goal distribution models to escaping UHRs and RRWs remains as future work.

Arvand-LS (Xie, Müller, and Holte 2014) is similar to Arvand, but it uses local GBFSs augmented with random walks to escape each UHR. Local best-first searches have also been useful for escaping UHRs in best-first search-based planners (Xie, Müller, and Holte 2015). Our results supplement this research by furthering our understanding of the differences between these competing ways to escape UHRs.

6 Conclusion

In this work, we theoretically and empirically analyzed two different methods for escaping UHRs: BrFS and RRWs. First, we characterized the expected runtime of these approaches for finding goals in a given search task, since for any UHR, we can always define a corresponding search task where finding a goal is equivalent to escaping from that UHR. In particular, we showed that RRWs will outperform BrFS in expectation if the success probability of a single random walk is proportional to the ratio of the number of states at the goal depth to the number of states shallower than the goal depth. In the case of trees, we refined this result by providing a lower bound on the density of goals at the goal depth for which RRWs are faster than BrFS.

Next, we considered RRW-based variants of EHC in order to compare the use of BrFS and RRWs for escaping UHRs in planning problems. We first showed that the existing worst-case runtime results for EHC can be extended to these RRW-based variants in the form of expected runtime guarantees. We then empirically compared EHC and the RRW-based variants on domains with different topological properties as a way to study the relative effectiveness of BrFS and RRWs for escaping UHRs. EHC was generally found to be better on bounded-UHR problems, while the RRW-based variants had better performance for unbounded-UHR domains.

Acknowledgments

We thank the anonymous reviewers for their helpful feedback on this work. We also wanted to thank Mark Loyzer and Allan Borodin for helping to initiate this line of research. Finally, we gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

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.

Everitt, T.; and Hutter, M. 2015. Analytical Results on the BFS vs. DFS Algorithm Selection Problem. Part I: Tree Search. In Pfahringer, B.; and Renz, J., eds., *AI 2015: Advances in Artificial Intelligence - 28th Australasian Joint Conference, Canberra, ACT, Australia, November 30 - December 4, 2015, Proceedings*, volume 9457 of *Lecture Notes in Computer Science*, 157–165. Springer.

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

Hoffmann, J. 2001. Local Search Topology in Planning Benchmarks: An Empirical Analysis. In Nebel, B., ed., *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, 453–458. Morgan Kaufmann.

Hoffmann, J. 2005. Where 'ignoring delete lists' works: local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24: 685–758.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

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

Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4): 173–180.

Nakhost, H.; and Müller, M. 2009. Monte-Carlo Exploration for Deterministic Planning. In Boutilier, C., ed., *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 1766–1771.

Nakhost, H.; and Müller, M. 2013. Towards a Second Generation Random Walk Planner: An Experimental Exploration. In Rossi, F., ed., *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2336–2342. IJCAI/AAAI.

Nakhost, H.; and Müller, M. 2014. Towards a theory of random walk planning: Regress factors, fair homogeneous graphs and extensions. *AI Communications*, 27(4): 329–344.

Platnick, D.; Tomasz, D.; Earl, E.; Khanzadeh, S.; and Valenzano, R. 2025. Breadth-First Search vs. Restarting Random Walks for Escaping Uninformed Heuristic Regions. arXiv:2511.09549.

Torralba, Á.; Seipp, J.; and Sievers, S. 2021. Automatic Instance Generation for Classical Planning. In Biundo, S.; Do, M.; Goldman, R.; Katz, M.; Yang, Q.; and Zhuo, H. H., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021*, 376–384. AAAI Press.

Xie, F.; Müller, M.; and Holte, R. 2014. Adding Local Exploration to Greedy Best-First Search in Satisficing Planning. In Brodley, C. E.; and Stone, P., eds., *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, 2388–2394. AAAI Press.

Xie, F.; Müller, M.; and Holte, R. 2015. Understanding and Improving Local Exploration for GBFS. In Brafman, R. I.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*, 244–248. AAAI Press.