# Anchor Search: A Unified Framework for Unbounded Bidirectional Search

by

Sepehr Mohammad Lavasani

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science
University of Alberta

# Abstract

In recent years, significant strides in optimal bidirectional heuristic search (Bi-HS) have deepened our theoretical understanding and boosted performance. Yet, algorithms for Bi-HS in unbounded suboptimal scenarios remains largely unexplored. Despite leveraging front-to-end (F2E) and front-to-front (F2F) bidirectional search for optimal algorithms, adapting them for unbounded suboptimal search remains an open challenge. We introduce a novel framework for suboptimal Bi-HS, called *anchor search*, and use it to derive new algorithms. Additionally, we propose using pattern databases (PDBs) as differential heuristics (DHs) to construct F2F heuristics—a necessity for F2F searches. Our experiments evaluate six anchor search algorithms across diverse domains, with a subset of them outperforming existing methods.

*To my beloved wife,*

*my parents,*

*and*

*my brother*

# Acknowledgements

Throughout my academic journey, I have been fortunate to work under the supervision of Dr. Nathan R. Sturtevant. I cannot express enough my appreciation for his invaluable guidance and support. His expertise and insightful feedback have been instrumental in shaping my ideas and enhancing my work. It has been an absolute pleasure and honor to have him as my supervisor.

I would also like to extend my sincere gratitude to Dr. Ariel Felner and Dr. Shahaf Shperberg for their valuable insights and suggestions that played a vital role in elevating the quality and depth of this study.

I am deeply grateful and appreciative of my wonderful wife, Anita Khalafbeigi, whose unwavering support, love, and understanding sustained me throughout this journey. Your encouragement fueled my determination, and your patience during late nights and weekends did not go unnoticed. This thesis is as much yours as it is mine, and I am endlessly grateful for having you by my side.

Finally, I want to extend my deepest gratitude to my friends, Shadan Golestan-Irani and Athar MahmoudiNejad whose presence added joy to my academic journey. Thank you for the countless conversations, shared laughter, and understanding during both the challenging and rewarding moments. Your presence has made this endeavor truly special.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Pathfinding is a type of *heuristic search* (HS) problem involving the task of finding a path between two states in a given state space. This is often guided by a heuristic function that estimates distances between states. Pathfinding has a wide variety of applications [1–5] in the real world, ranging from navigation in robotics and video games to finding a sequence of actions leading to solving of a puzzle. Although finding the shortest path rather than a suboptimal solution is often crucial and preferable in many scenarios, real-time constraints and limited computational resources can make optimal pathfinding impractical in many applications [6, 7]. For instance, in video games, a path must be found in a fraction of a second to maintain the target framerate on computers and gaming consoles. Additionally, pathfinding problems exist with enormous state spaces for which finding the optimal solution can take days or weeks, even using high-end computing resources [2]. Hence, suboptimal pathfinding becomes important as finding a suboptimal path is better than having no solution due to exhausting the available resources while trying to find the optimal path.

From the optimality lens, pathfinding problems can be divided into three main classes: optimal HS, unbounded suboptimal HS, and bounded suboptimal HS. Optimal HS aims to find the shortest path between the start and goal states, while unbounded suboptimal HS accepts any arbitrary length path between the ends. Bounded suboptimal HS strikes a middle ground between optimal and unbounded

suboptimal HS by imposing an upper bound upon the solution length relative to the least-cost path. In numerous scenarios, achieving optimal solutions is impractical due to the extensive computational requirements [8], leading to a preference for using unbounded suboptimal HS.

Existing HS algorithms typically adopt one of two main approaches to search the search space: unidirectional HS (Uni-HS), and bidirectional HS (Bi-HS). Uni-HS algorithms search for a path from *start* toward *goal* while being guided by a heuristic function that estimates the cost of reaching *goal* from any other states in the graph. Alternatively, Bi-HS involves simultaneous searching from both *start* (forward search) and *goal* (backward search) until the two frontiers meet.

Based on the heuristic being used, the study of Bi-HS algorithms primarily divides into two categories: front-to-end (F2E) [9] and front-to-front (F2F) algorithms [10]. In F2E Bi-HS, heuristic functions estimate the distances from individual states to either *start* or *goal* while F2F algorithms employ heuristics that estimate the distances between any arbitrary pair of states.

## 1.1 Brief History of Bidirectional Search

In this section, we will give a short overview of the history of bidirectional search. We will discuss the advancements that highlight the potential of bidirectional search, as well as the gaps that reveal areas of current and future research. This historical background will provide the necessary context for establishing the foundation and motivation behind our contributions.

The history of bidirectional search dates back to 1966, when Nicholson [11] introduced *bidirectional search* (BS). BS involves two simultaneous searches, one starting at the start and the other at the goal. In the absence of heuristics, BS demonstrated superior performance over Dijkstra's algorithm (DA) [12], an uninformed unidirectional search algorithm, in terms of expansions and time.

Intuitively, in a 2-dimensional polynomial domain where the number of states at

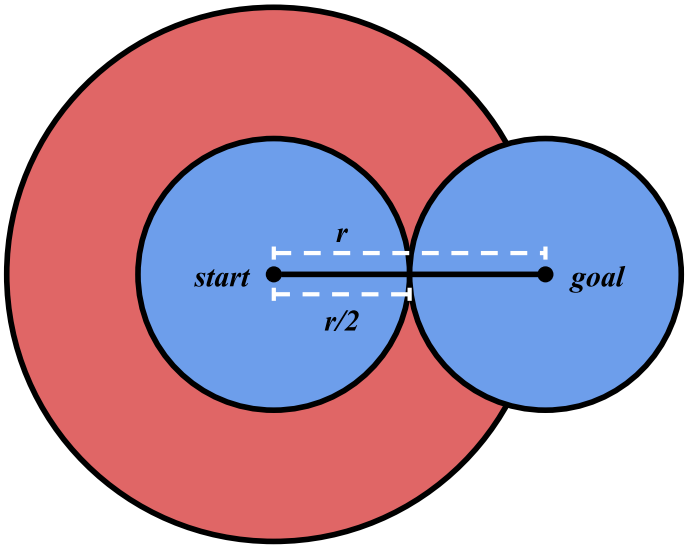



Figure 1.1: The area expanded by DA (red circle) and BS (blue circles).

depths smaller than $d$ grows polynomially with $d$, the expansions performed by DA can be visualized as a continuously expanding solid circle centered at the start. This circle keeps growing until it reaches the goal located in distance $r$ of the start (the red circle in Figure 1.1). In this case, the area covered by DA is $\pi r^2$. Applying the same analogy to BS, we can imagine two solid circles centered at the start and the goal (blue circles in Figure 1.1). These circles grow until they collide. In this scenario, the combined area covered by both circles would be $2 \times \pi(\frac{r}{2})^2 = \pi r^2/2$ – half of the area expanded by a single instance of DA. In an exponential domain with a branching factor of $b$, the number of states expanded by DA to reach a goal state at depth $d$ from the start state is approximately $b^d$. On the other hand, employing BS results in roughly $2 \times b^{(d/2)}$ expansions, which is exponentially fewer than the expansions performed by DA.

In 1968, coupled with admissible heuristics, A* [13] was introduced and later shown to dominate any admissible best-first search algorithm, including DA [14]. The introduction of the concepts of bidirectional search and heuristic search paved the way for new possibilities, leading to the question of whether combining these two approaches would result in a promising option. However, the first algorithm that attempted to

do so, BHPA [15], did not yield significant gains despite the potential of both ideas when used independently. Pohl conjectured that BHPA was ineffective because frontiers usually missed each other and met near either the start or goal states [15]. Sint and De Champeaux [10] proposed BHFFA to address this shortcomings by adopting F2F heuristics to encourage frontiers to meet in the middle. Although using F2F heuristics helped address the issue of frontiers missing in BHPA, and resulted in fewer expansions compared to a generalized version of A$^*$, the F2F heuristics were computationally expensive, which made the BHFFA algorithm significantly slower in terms of runtime when compared to both Uni-HS and F2E Bi-HS algorithms of that time.

For years, researchers have sought to understand why Bi-HS falls short in comparison to Uni-HS. Nilsson [16] highlighted Pohl's explanation: the frontiers can miss each other, meaning they might meet close to either the start or goal states. This occurrence, referred to as the frontiers missing or crossing, implies that the progress made by one of the frontiers do not contribute to finding the path. In such situations, employing a unidirectional algorithm could potentially yield a path with less computational effort. Later, Kaindl and Kainz [9] provided evidence contrary to Nilsson's explanation: the frontiers meet, but the substantial effort required to prove optimality puts Bi-HS at a disadvantage compared to Uni-HS. Finally, Barker and Korf [17] suggested that in most cases, either Uni-HS or bidirectional brute-force search outperform Bi-HS. However, this theory was based on assumptions that are often not realistic, such as the assumption that solving a problem is equally difficult in both directions and that the frontiers meet exactly in the middle of the path. At the time, no algorithm had achieved this.

Inspired by the above theories on the shortcomings of Bi-HS [9, 17], recent years have witnessed significant advancements in this area of research, challenging conventional understanding. The first advancement was the introduction of MM [18], a Bi-HS algorithm ensuring that the search frontiers meet at the midpoint of the optimal path, an assumption from Barker and Korf's theory. Shaham *et al.* [19] in-

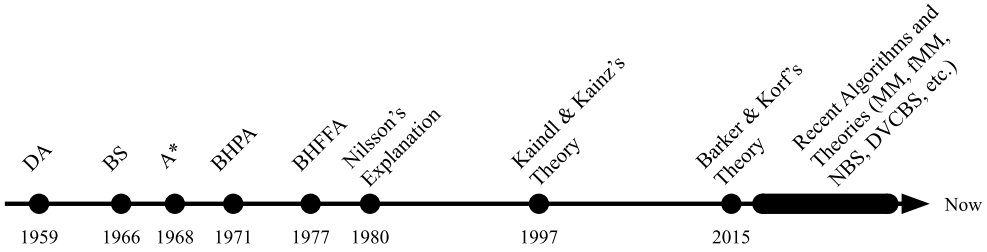



Figure 1.2: Timeline of milestones in bidirectional search.

troduced fMM, an extension of MM that allows for parameterizing the fraction of the path constructed by the forward search. Subsequently, identifying the theory of sufficient conditions for bidirectional search [20] paved the way for a better understanding of when Bi-HS outperforms Uni-HS [21]. Shaham *et al.* [19] expanded the theory by demonstrating that fMM can solve problems only by performing the necessary expansions when prior knowledge guided the choice of its parameter value. Chen *et al.* [22] translated the sufficient conditions for bidirectional search into a vertex cover problem on a bipartite graph, known as a *must-expand graph* ($G_{MX}$). Building upon this insight, two Bi-HS algorithms were proposed: the Near-Optimal Bidirectional Search (NBS) algorithm [22], which expands at most twice the minimum required nodes to reach the optimal solution, and DVCBS [23], guided by finding smallest vertex cover during search.

Figure 1.2 visualizes the chronological progression of bidirectional search. To conclude, Barker and Korf's conjecture [17] sparked a notable interest in optimal Bi-HS in recent years, resulting in refined theoretical understanding [20, 21, 23, 24] and the development of efficient algorithms [18, 22, 23]. These advancements include an in-depth theoretical and empirical comparison between unidirectional and bidirectional search methodologies [21].

## 1.2 Motivation

This thesis addresses the problem of unbounded suboptimal Bi-HS. Despite the extensive studies conducted on optimal and suboptimal Uni-HS in the past [14, 25–28] and recent advancements in optimal Bi-HS, suboptimal Bi-HS has not received the same level of investigation, particularly when it comes to unbounded suboptimal Bi-HS, which has remained largely unexplored. This lack of exploration raises many questions about the best algorithms and heuristics to use in this context. There is no theory that explains the conditions under which suboptimal Bi-HS outperforms suboptimal Uni-HS. Additionally, we have not identified the heuristic properties that measure how suitable a given heuristic is for suboptimal or greedy search. This dissertation provides new insights into suboptimal Bi-HS algorithms and how to build front-to-front heuristics.

## 1.3 Key Contributions

This thesis introduces the *anchor search* framework, synthesizing various design choices for unbounded suboptimal Bi-HS algorithms that utilize both F2F and F2E heuristics. We show that existing unbounded suboptimal Bi-HS algorithms, such as $D$-node retargeting [29] and the bidirectional version of greedy best-first search, can be viewed as special cases of this framework. Additionally, we introduce new promising algorithms, such as *temporal* and *randomized anchor search*, within this framework.

This thesis also addresses a previously overlooked question – the construction of F2F heuristics, especially in domains like the Towers of Hanoi (TOH) where strong F2F heuristics are not readily available. Along these lines, this thesis studies the possibility of using pattern database (PDB) heuristics [30] as differential heuristics (DHs) [31] to establish the state-to-state heuristics required for anchor search.

## 1.4 Thesis Outline

The thesis consists of six chapters. In Chapter 2, "Background and Related Work," we discuss the relevant literature to provide essential context for unbounded suboptimal bidirectional heuristic search, and identify gaps in existing approaches. Chapter 3, "Anchor Search," provides a comprehensive description of the anchor search framework and discusses how existing algorithms fit into this framework. It also presents promising design choices used to propose novel anchor search algorithms. In Chapter 4, "Front-to-Front Heuristics," we provide details of our front-to-front heuristics approaches, which are required for anchor search. Chapter 5, "Results," presents experimental findings across various domains to highlight the effectiveness of anchor search. Finally, Chapter 6, "Conclusion and Future Work," synthesizes our key findings, discusses implications, and suggests future research directions.

# Chapter 2

# Background and Related Work

There has been notable recent progress in both the theory [20, 32] and algorithms [22, 23, 33] for optimal Bi-HS. This thesis focuses on unbounded suboptimal search, an area that has received comparatively less attention. In this chapter, we provide background information on this topic, encompassing the problem definition and the algorithms that this thesis builds upon.

## 2.1 Problem Statement and Definitions

This thesis addresses the problem of finding a path of arbitrary length between two states (vertices) within a graph, referred to as *start* and *goal*. The problem input consists of a graph $G = (V, E)$, with $V$ and $E$ representing vertices and edges, a cost function $c : E \rightarrow \mathbb{R}^+$ assigning costs to edges, and a heuristic function $h(u, v) : V \times V \rightarrow \mathbb{R}+$, estimating distances between state pairs. It is important to note that in unbounded suboptimal search, admissibility and consistency of heuristics are not required [34, 35]. The output is a path $\pi = v_0, ..., v_n$ with a cost of $\sum_{i=0}^{n-1} c(v_i, v_{i+1})$, where $v_i, v_{i+1} \in E$. Paths are evaluated with respect to their length.

All algorithms discussed in this thesis are assumed to be expansion-based [14, 20]. At any given *iteration*, the process through which a single state is selected and expanded, the $g$-cost of an arbitrary state $s$ denotes the cost of the shortest path found thus far between $s$ and its corresponding frontier's origin (start/goal). For

an arbitrary pair of states $a$ and $b$, $d(a, b)$ represents the cost of the optimal path between $a$ and $b$. $D$ and $\bar{D}$ are used as subscripts to differentiate symbols for the current and opposite direction of search. Notably, states and nodes are equivalent notions throughout this dissertation.

## 2.2 Suboptimal Bidirectional Search

*Front-to-end* (F2E) and *front-to-front* (F2F) heuristics are two main types of heuristics that Bi-HS algorithms have used. In a state space $V$, a F2E heuristic $h_{F2E} : V \rightarrow \mathbb{R}^+$ estimates the distance from any given state $s \in V$ to the start or goal states. In contrast, a F2F heuristic $h_{F2F} : V \times V \rightarrow \mathbb{R}^+$ evaluates any arbitrary state pair $(s_1, s_2) \in V \times V$. When performing full F2F evaluations, evaluating a given state $s$ in the forward (or backward) frontier involves taking the minimum of $h_{F2F}(s, s')$, for all states $s'$ in the backward (or forward) frontier in order to establish a lower bound on the estimated distance of $s$ to the entire opposite frontier. Although this is a more informative evaluation compared to what F2E heuristics offer, it comes at the cost of evaluating many pairs of states. Therefore, despite the fact that, generally, F2F Bi-HS algorithms expand fewer states, the expensive computational cost of each expansion makes them impractical in terms of runtime [36].

To address the large time complexity of pure F2F evaluations, Politowski and Pohl [29] introduced the concept of a $d$-node: a dynamic representative state in each frontier, which serves as a temporary goal for the opposite frontier. The primary purpose of $d$-nodes was to facilitate frontier convergence while circumventing the computational burden of performing a complete F2F evaluation for all pairs of frontier states. However, $d$-node retargeting (DNR), the Bi-HS algorithm which introduced $d$-nodes, requires that the open list of the opposite frontier is re-sorted whenever a new $d$-node is chosen. Kuroiwa and Fukunaga [37] proposed top-to-top bidirectional search (TTBS) as a modification of DNR, seeking to mitigate the considerable overhead linked with repeatedly sorting the open list.

Mayer and Krebsbach [38] provided definitions for possible variations of greedy and optimal unidirectional and bidirectional searches. Their empirical results suggest that bidirectional search could outperform its unidirectional rivals in terms of state expansions. However, their comparison lacked the inclusion of state-of-the-art Bi-HS algorithms of the time and their analysis was restricted to the sliding tile puzzle with varying sizes, with the number of expansions as the only measure of comparison.

Two recent papers have shown the potential effectiveness of suboptimal Bi-HS search over unidirectional search in practice. A*-connect [39] is a bounded suboptimal bidirectional algorithm that outperforms state-of-the-art unidirectional algorithms in motion planning. A*-connect utilizes the notion of pivots, which resemble $d$-nodes. Pivots are a set of representative states encapsulating the search frontier. In other work, Atzmon *et al.* [40] studied how the ideas behind Weighted A* [41] can be applied to the MM algorithm [18], giving Weighted MM (WMM). This is the first step in extending the theory of optimal Bi-HS to bounded suboptimal Bi-HS.

## 2.3 Baselines

In order to conduct a comprehensive evaluation of anchor search's efficacy, we include unbounded suboptimal Uni-HS, F2E Bi-HS, and F2F Bi-HS algorithms as the baselines of this work. The chosen baselines are described in detail in the remainder of this section.

### 2.3.1 Greedy Best-First Search (GBFS)

GBFS [42], also known as pure heuristic search, is a best-first, unidirectional search algorithm that expands the state with the minimum $h$-value. GBFS is greedy with respect to the heuristic, as it does not consider the cost of reaching a state ($g$-cost) when evaluating it for expansion. Given its simplicity and wide use in planning [37, 43, 44], GBFS is a clear choice for baseline comparisons.

### 2.3.2 Bidirectional GBFS (BGBFS)

GBFS can be utilized bidirectionally, known as BGBFS, providing an additional potential baseline for this research. BGBFS uses two alternating GBFS searches, one from the start towards the goal and vice versa. In other words, at each step, the algorithm expands the state $s \in Open_D$ with the smallest heuristic value $h_D(s)$ towards the opposite end and then switches the current direction $D$. The search is terminated once one frontier generates a state that belongs to the opposite frontier's open list. Despite the simplicity of this algorithm, it has only been briefly discussed in the literature [38]. The pseudocode of BGBFS is provided in Algorithm 1. Note that $h_f$ and $h_b$ are front-to-end heuristics used in forward and backward frontiers, respectively.

Even though the literature does not contain information about the behavior of BGBFS, particularly in practice, we can still borrow some theoretical justifications from other studies suggesting the potential effectiveness of BGBFS. Running two simultaneous GBFS instances can be considered a limited realization of a parallelization technique called *dovetailing* [45]. Dovetailing enables us to run several configurations of a parameterized algorithm in parallel when we have no prior information about the best problem-specific tuning. Given that we are finding a path between a pair of states in an undirected graph, we can take either state as the start and the other as the goal state. Thus, choosing one of the two endpoints as the start would be a parameter needed to be tuned when we aim to solve a problem unidirectionally. From this perspective, we can utilize dovetailing by running the search simultaneously in both directions, as we do not know the best direction in advance. In the worse case, when the frontiers meet close to the ends, BGBFS with the alternating policy will expand at most twice the states expanded by unidirectional GBFS in the optimal direction — the direction that minimizes the expansions performed by GBFS.

**Algorithm 1** BGBFS

**Input:** $G, start, goal, h_f, h_b$

▷ $h_f$ and $h_b$ are forward and backward heuristics.

**Output:** a path between *start* and *goal*

```
1: D ← forward
2: add start to Open_D
3: add goal to Open_D̄
4: while Open_D is not empty do
5:     next ← argmin_{n∈Open_D} h_D(n)
6:     remove next from Open_D
7:     add next to Closed_D
8:     for s ∈ Successors(next) do
9:         if s ∉ Closed_D then
10:            add s to Open_D
11:        end if
12:        if s ∈ Open_D̄ then
13:            return ExtractPath(start, goal)
14:        end if
15:    end for
16:    D ← D̄                          ▷ Switch the current direction.
17: end while
18: return null
```

### 2.3.3 $d$-node Retargeting (DNR)

DNR was one of the first F2F Bi-HS algorithms that attempted to guide the frontiers together. It mitigated the computational overhead of full F2F evaluations by maintaining a dynamic representative state, known as a $d$-node, making it a fundamental baseline in our study. In DNR, as shown in Algorithm 2, each frontier performs $k$ consecutive expansions prioritized by $f(n) = (1 - w)g(n) + wh(n, dnode_{\bar{D}})$ with respect to the $d$-node in the opposite frontier; we call this a *turn*. Then, the state with the largest $g$-cost in the current frontier becomes the next $d$-node. Once the $d$-node is updated, the opposite frontier must sort its entire open list based on $f$ with respect to the newly updated $d$-node before taking its turn. The implementation of DNR used in our study uses $w = 1$, with ties being broken in favor of the state with a higher $g$-cost.

The optimal value of $k$ in DNR depends on the actual solution length, which is not known in advance. A larger value of $k$ results in performance similar to unidirectional best-first search algorithms, while a smaller value of $k$ results in more frequent retargeting and poorer runtime performance.

### 2.3.4 Top-to-top Bidirectional Search (TTBS)

**TTBS** (Algorithm 3), the final baseline algorithm, takes a different approach to the concept of $d$-nodes. It operates on the assumption that the highest priority states in the opposite open lists should ideally be close to each other. Unlike DNR, which involves re-sorting the entire open list every time a $d$-node changes, TTBS approximates this process during each expansion through a lazy evaluation of a subset of the open list. TTBS uses the top (the highest priority state) of each open list as the $d$-node for that specific frontier. Algorithm 3 describes TTBS, where $top_D$ and $top_{\bar{D}}$ correspond to the tops of the current and opposite frontiers' open lists, respectively. States in $Open_D$ are primarily prioritized based on their $h$-cost to $top_{\bar{D}}$. In the process of selecting a state to expand, TTBS examines if $top_{\bar{D}}$ (the opposite $d$-node)

**Algorithm 2** DNR

**Input:** $G, start, goal, h, k$
**Output:** a path between *start* and *goal*

1: $D \leftarrow forward$
2: add *start* to $Open_D$
3: add *goal* to $Open_{\bar{D}}$
4: $dnode_D \leftarrow start$
5: $dnode_{\bar{D}} \leftarrow goal$
6: **while** no path has been found **do**
7:  **for** $i = 1, ..., k$ **do**
8:  **if** $Open_D$ is empty **then**
9:  **return** null
10:  **end if**
11:  $next \leftarrow \underset{n \in Open_D}{\operatorname{argmin}} f(n)$
12:  remove *next* from $Open_D$
13:  add *next* to $Closed_D$
14:  **for** $s \in$ SUCCESSORS(*next*) **do**
15:  **if** $s \notin Closed_D$ **then**
16:  add $s$ to $Open_D$
17:  **end if**
18:  **if** $s \in Open_{\bar{D}}$ **then**
19:  **return** EXTRACTPATH(*start*, *goal*)
20:  **end if**
21:  **end for**
22:  **end for**
23:  $furthest \leftarrow \underset{n \in Open_D}{\operatorname{argmax}} g_D(n)$
24:  **if** $g(furthest) > g(dnode_D)$ **then**
25:  $dnode_D \leftarrow furthest$
26:  RETARGET($Open_{\bar{D}}$, $dnode_D$)
27:  **end if**
28:  $D \leftarrow \bar{D}$ ▷ Switch the current direction.
29: **end while**

| Optimality | Uni-HS | Bi-HS | |
|---|---|---|---|
| | | **F2F** | **F2E** |
| **Optimal** | A* | BHFFA | BHPA, MM, NBS, DVCBS |
| **Bounded suboptimal** | Weighted A* | A*-connect | WMM |
| **Unbounded suboptimal** | GBFS | DNR, TTBS | BGBFS |

Table 2.1: Classification of heuristic search algorithms.

is *similar* to $T(top_D)$: the $d$-node that $top_D$ was last prioritized against, where two states are identified as *similar* if they are equal or adjacent (in undirected graphs). If such similarity is identified, $top_D$ is expanded. Conversely, if no similarity is found, $top_D$ is *reprioritized* against the current $top_{\bar{D}}$. This process continues until a state is expanded. Note that, while prioritizing a state, if there are multiple states with the same heuristic value priority, those that have already been reprioritized earlier hold higher priorities to prevent repeated reprioritization for a given state. Additionally, FIFO and LIFO strategies can be employed for further tie-breaking.

In the context of unbounded suboptimal heuristic search, we have identified four noteworthy algorithms, each with their strengths and weaknesses that capture diverse heuristic search approaches. GBFS stands as a straightforward representative of unbounded suboptimal Uni-HS algorithms. Despite its proven robustness, it faces a limitation in its inability to leverage F2F heuristics when available. BGBFS, a bidirectional variant of GBFS, serves as a test case to explore whether being bidirectional alone can enhance performance compared to unidirectional greedy search. Finally, DNR and TTBS embody two realizations of unbounded suboptimal F2F Bi-HS in our study. While they align more closely with the assumptions of planning problems, their inclusion allows for a more comprehensive understanding of unbounded suboptimal Bi-HS in practice, even considering their demanding operations such as retargeting in DNR and maintaining the lazy priority queue in TTBS.

**Algorithm 3** TTBS

**Input:** $G, start, goal, h, k$
**Output:** a path between *start* and *goal*

```
1: D ← forward                                  ▷ Initialize direction D with forward.
2: push(Open_D, start)
3: push(Open_D̄, goal)
4: T(start) ← goal
5: T(goal) ← start
6: while no path has been found do
7:     for i = 1, ..., k do                     ▷ Perform k expansions in each direction.
8:         if Open_D is empty then
9:             return null
10:        end if
11:        next ← pop(Open_D)                   ▷ Pop the highest priority state from Open_D
12:        while T(next) is not similar to top_D̄) do
13:            push(Open_D, next)               ▷ Insert next in Open_D with new priority.
14:            T(next) ← top_D̄
15:            next ← pop(Open_D)
16:        end while
17:        add next to Closed_D
18:        for s ∈ Successors(next) do
19:            if s ∉ Closed_D then
20:                push(Open_D, s)
21:            end if
22:            if s ∈ Open_D̄ then
23:                return ExtractPath(start, goal)
24:            end if
25:        end for
26:    end for
27:    D ← D̄                                    ▷ Switch the current direction.
28: end while
```

# Chapter 3

# Anchor Search

In general, F2F heuristics are more informed than F2E ones [36]. However, the quadratic complexity of assessing all pairs of frontier states required by optimal search [20] makes full F2F evaluations expensive [36]. F2F Bi-HS algorithms can reduce this overhead by giving up the requirement of finding optimal solutions. With this relaxation in mind, we introduce the anchor search framework, which unifies multiple approaches to suboptimal F2F search into a single framework.

The anchor search framework revolves around two fundamental concepts: the concepts of an *anchor* and *candidates*, illustrated in Figure 3.1. An *anchor* is a representative state associated with each frontier, treated as a provisional goal by the opposite frontier. The anchor notion is broader than the $d$-node notion in the sense that anchors are arbitrary states that do not necessarily have to be part of their corresponding frontier. Anchor search is detailed in Algorithm 4. Each iteration of the algorithm involves selecting the next state for expansion from a set of *candidate* states $C$, extracted from the current frontier's open list ($Open_D$), as depicted in line 2. The best candidate is the state $c \in C$ with the lowest heuristic value $h(c, anchor_{\bar{D}})$, i.e., the closest candidate to the anchor of the opposite frontier $anchor_{\bar{D}}$ based on the heuristic estimation (line 3). Subsequent to expanding a state from the current frontier, the anchor search may update $anchor_D$ and/or alter the current direction $D$ as guided by the anchor selection and direction selection policies (lines 18 and 19). In

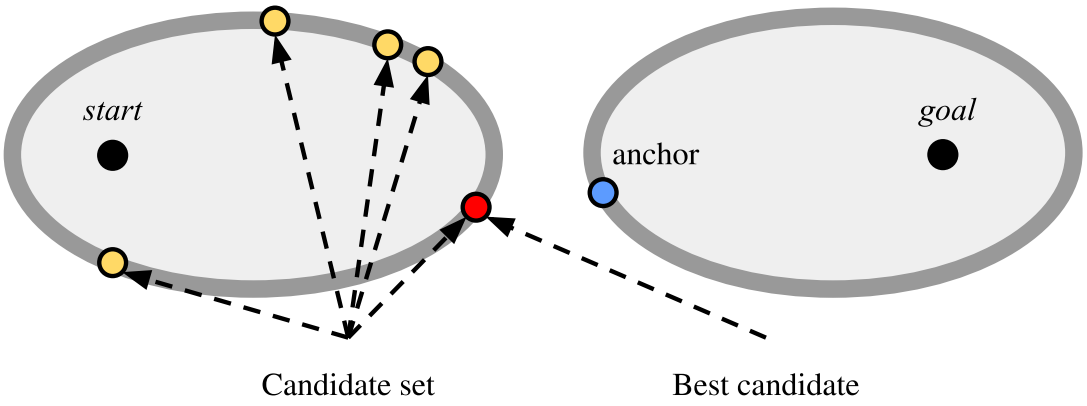



Figure 3.1: Anchor search illustration; the blue dot represents the backward frontier's anchor, the red dot shows the next state to expand, and the yellow dots are other candidates.

Algorithm 4, the GetAnchor, GetCandidates, and GetDirection procedures correspond to the anchor selection, candidate selection, and direction selection policies that are required to be specified for any specific instance of anchor search. In the next sections, we introduce a variety of distinct anchor and candidate selection policies.

## 3.1 Anchor Selection

The anchor selection policy governs the process of selecting the anchor from all the states generated so far during each iteration. No universal policy exists for choosing an anchor, as the best selection can be influenced by factors such as the problem domain, the heuristic, and other selection policies (candidates and direction). Nonetheless, we introduce a range of anchor selection policies that have demonstrated utility in our empirical study.

- **Temporal:** The anchor is the most recently expanded state in each direction.
- **Closest to the goal:** The anchor is the state $s$ among all the states generated so far that minimizes $h(s, goal_D)$.
- **Closest to the opposite anchor:** The anchor is updated to the most recently

**Algorithm 4** Anchor Search

**Input:** $G, start, goal, h$
**Output:** a path between *start* and *goal*

```
1: while Open_D is not empty do
2:     C ← GetCandidates(Open_D)
3:     next ← argmin_{n∈C} h(n, anchor_D̄)
4:     remove next from Open_D
5:     add next to Closed_D
6:     for s ∈ Successors(G, next) do
7:         if s ∉ Closed_D then
8:             add s to Open_D
9:             parent(s) ← next
10:        else if g(s) > g(next) + c(next, s) then
11:            g(s) ← g(next) + c(next, s)
12:            parent(s) ← next
13:        end if
14:        if s ∈ Open_D̄ then
15:            return ExtractPath(s, start, goal)
16:        end if
17:    end for
18:    anchor_d ← GetAnchor(Open_D, Closed_D)
19:    D ← GetDirection()
20: end while
21: return null
```

expanded state $s$ if $h(s, anchor_{\bar{D}})$ is smaller than $h(anchor_D, anchor_{\bar{D}})$.

- **Random:** The anchor is randomly selected from the open/closed list.
- **Fixed to the origin:** The anchor is the *start*/*goal* state and never changes.

It is important to note that the anchor selection policy can also outline the frequency of anchor updates, either explicitly or implicitly. This flexibility facilitates the design of both F2E and F2F Bi-HS algorithms. Should the anchor remain unchanged from the frontier's origin, the resulting algorithm aligns with F2E search. In scenarios where the anchor can vary, the algorithm aligns with F2F search.

## 3.2 Candidate Selection

Anchor search assembles a set of candidates, among which the next state for expansion is selected. It prioritizes states according to their $h$-cost relative to the opposite anchor. However, unlike optimal search algorithms, it is not necessary to involve the entire open list in this prioritization. Consequently, for any given instance of anchor search, it is necessary to define which subset of the open list to consider during each expansion by defining the candidate selection policy. The candidate-selection policies we consider are as follows:

- **Brute-force:** In the most trivial way, candidates are all states on the open list.
- **Temporal:** Candidates are the set of most recently generated states.
- **Random:** Candidates are a subset of states selected randomly from the open list.

## 3.3 Direction Selection

The direction selection policy determines the expansion direction in each iteration of the algorithm. In its simplest form, the *alternating* policy toggles the expansion direction with every iteration. In contrast, the $n$-alternating policy switches the expansion direction every $n$ iterations. This direction change can be contingent on conditions such as the maximum or minimum $h$-cost within a frontier. Furthermore, it is possible to craft Uni-HS algorithms in the anchor search framework by maintaining the same expansion direction throughout the search. Within the scope of this study, all novel anchor search algorithms adopt the alternating policy.

## 3.4 Completeness of Anchor Search

Anchor search expands the best state among the candidates with respect to the opposite anchor and never re-expands any state. Consequently, it functions as a best-

first search employing a complex priority function. Chen and Sturtevant [24] showed that a best-first search utilizing any priority function is always complete in finite state spaces in the absence of re-expansions. Thus, algorithms within the anchor search framework are complete, regardless of the specific candidate selection, anchor update, and direction choice policies employed. It is worth mentioning that anchor search does not allow evicting any state from the open list unless that state is expanded, ensuring the framework's completeness.

## 3.5 Variants

Enumerating all possible algorithms that can be devised from the anchor search framework is impractical within the scope of this thesis, as not all variants are viable or have significant promise. Therefore, we will focus on promising variants and those that showed reasonable performance in initial, more broad experiments. Our analysis will involve categorizing algorithms according to their anchor and candidate selection policies, while also examining the impact of different heuristic types.

Our naming convention in this thesis refers to a specific anchor search algorithm as $\mathrm{AS}_{\beta}^{\alpha(k)}$, where $\alpha$ denotes the candidate selection policy, $k$ specifies the number of candidates, and $\beta$ indicates the anchor selection policy. Considering that the candidate selection policy involves important data structure considerations, we will initially divide anchor search algorithms into three primary classes: *brute-force anchor search* ($\mathrm{AS}^{B}$), *temporal anchor search* ($\mathrm{AS}^{T}$), and *randomized anchor search* ($\mathrm{AS}^{R}$), regardless of the chosen anchor selection policy. To denote the anchor selection policy, a subscript will be employed. For instance, $\mathrm{AS}_{T}^{T}$, $\mathrm{AS}_{A}^{T}$, and $\mathrm{AS}_{F}^{T}$ correspond to $\mathrm{AS}^{T}$ with temporal (T), closest-to-the-opposite-anchor (A), and fixed-to-the-origin (F) anchor selection policies, respectively. Additionally, an anchor search algorithm may utilize different anchor selection policies for the forward and backward frontiers, referred to as hybrid variants. For instance, $\mathrm{AS}_{AF}^{T}$ represents an $\mathrm{AS}^{T}$ algorithm where the forward frontier adopts the closest-to-the-opposite-anchor policy, while the backward

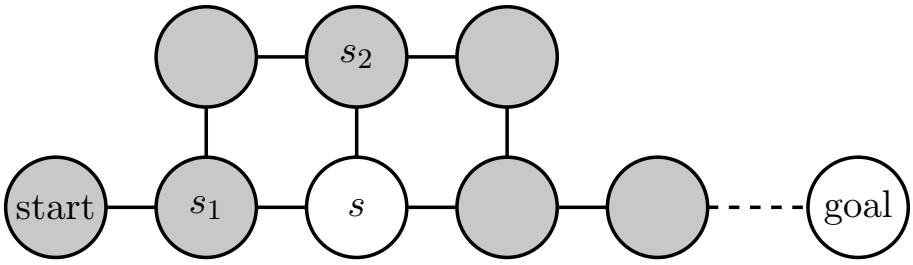
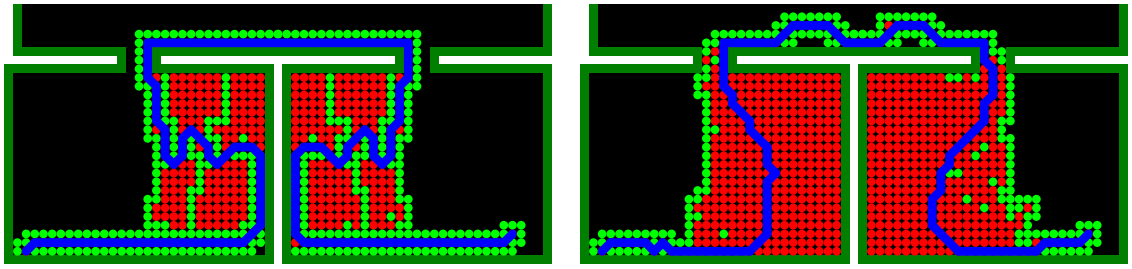



Figure 3.2: States highlighted in gray depict the forward frontier's closed list as well as the part of the path found in the forward direction by $\text{AS}^T$ without expanding $s$.



(a) Without reprioritization (b) With reprioritization

Figure 3.3: The impact of reprioritization on solution quality of $\text{AS}^T_T$

frontier uses the fixed-to-the-goal policy. Table 3.1 classifies the proposed anchor search algorithms according to the anchor and candidate selection policies they employ.

### 3.5.1 Brute-force Anchor Search ($\text{AS}^B$)

The most straightforward approach for generating candidates is to consider the whole open list, that is, evaluating all the states in the open list and expanding the state $s \in C$ with the lowest $h(s, anchor_{\bar{D}})$. However, maintaining the open list in a sorted order based on $h$-cost becomes impractical when anchors change frequently (e.g., $\text{AS}^B_T$, $\text{AS}^B_R$, and $\text{AS}^B_A$). Therefore, without further assumptions or optimizations, identifying the most promising state for expansion would require a linear traversal of the open list. This relatively high time complexity renders such $\text{AS}^B$ variants infeasible, particularly as the problem complexity and, consequently, the size of the open list increases.

### 3.5.2 Temporal Anchor Search ($\text{AS}^T$)

We introduce a subclass of anchor search variants, called *temporal anchor search*, where candidate states are selected according to the order in which they were generated. Specifically, $\text{AS}^T$ focuses solely on the $k$ most recently appended states in the open list for constructing the candidate set (for some parameter $k$). During the expansion phase, the algorithm selects the most promising state among the top $k$ candidates in the open list and expands it. Notably, all $\text{AS}^T$ algorithms in our study prioritize states with a higher $g$-cost for tie-breaking.

We observed that $\text{AS}^T$ algorithms often find circuitous paths, as demonstrated in Figure 3.3(a). Figure 3.2 pinpoints where this phenomenon happens. As shown in Figure 3.2, the algorithm would have found a shorter solution if state $s$ had been expanded during the search. States such as $s$ can be identified once re-generated with a higher $g$-cost while already on the open list. The subsequent list of criteria generalizes the phenomenon depicted in Figure 3.2, describing the relationship among states $s_1$, $s_2$, and $s$:

1. State $s$ has been re-generated by $s_2$, while already existing on the open list, with $s_1$ as its current parent.

2. States $s_1$ and $s_2$ have been expanded in the same direction with $g$-costs of $g_1$ and $g_2$, respectively.

3. $g_2 - g_1 > c(s_1, s) + c(s, s_2)$, where $c$ is the cost function.

4. The search is terminated before expanding $s$ and both $s_1$ and $s_2$ are part of the final solution.

Thus, to mitigate this phenomenon, if a state on the open list is re-generated during the search with a higher $g$-cost, we *reprioritize* it as a recently generated state. This is the opposite of $\text{A}^*$ search, where a state is reprioritized when a smaller $g$-cost is found. Figure 3.3 (b) shows the impact of reprioritization in grid pathfinding.

**Data structures**

$\text{AS}^T$ involves two considerations related to data structures. Firstly, the temporal order of states added to the open list should be preserved. In our implementation, when removing a state $s$ from the open list (represented as a vector), we swap $s$ with the top element of the open list and then perform a pop operation to remove it (this applies to reprioritization as well). While this approach may not perfectly preserve the exact temporal order of the open list, it eliminates the need to shift elements after removal, making the data structure much faster than a heap and approximating the desired behavior quite closely. Secondly, a hash table can be utilized to efficiently retrieve the position of any given state in the open list (required for duplicate detection when states are regenerated).

### 3.5.3 Randomized Anchor Search ($\text{AS}^R$)

In this class of variants, *randomized anchor search*, the candidate set $C$ consists of a subset of $k$ states randomly sampled from the open list, either with or without replacement. Therefore, the open list requires no maintenance before or after expansion, resulting in each expansion taking time proportional to $k$. If the state $s^*$ with the lowest $h(s, anchor_{\bar{D}})$ is always included as a candidate, then $\text{AS}^R$ would be equivalent to $\text{AS}^B$ in terms of expansions. The likelihood of $s^*$ being sampled increases proportionally to $k$.

In an ideal scenario with no local minimum in the heuristic, $s^*$ in the current iteration would be among the successors of the state expanded in the previous iteration of the same frontier [46]. If the problems given to the algorithm are mostly in line with this ideal scenario, including the generated state with minimum $h$-cost from the previous iteration of a frontier in $C$ increases the chance of $s^*$ being sampled, and $\text{AS}^R$ achieves the same performance with smaller values of $k$. Thus, after this refinement, $C$ would consist of $k-1$ states chosen at random, along with the most promising generated state from the previous iteration.

## 3.6 BGBFS, DNR, and TTBS as Anchor Search Variants

$\text{AS}^B$ variants become more viable when the anchors remain fixed or change less often during the search. Although finding the state with the minimum $h$-cost with respect to a constantly changing anchor is less expensive than full F2F evaluations, its linear time complexity with respect to the frontier's size makes it impractical as the problem size grows. By employing the fixed-to-the-goal anchor selection policy in $\text{AS}^B$, the resultant algorithm, named $\text{AS}^B_F$, essentially mirrors BGBFS. In this scenario, the worst-case complexity of maintaining the sorted open list after each insertion or deletion would be $O(\log(n))$, where $n$ represents the size of the open list.

Similarly, DNR can be considered a variant of $\text{AS}^B$ where, following every $n$ iterations, the direction is switched, and the $d$-node (anchor) is updated to the state with the maximum $g$-cost. When $d$-nodes change infrequently, a complete re-sorting (retargeting) of the open list is necessary only upon $d$-node updates. DNR distributes the cost of retargeting over subsequent expansions when the anchor remains constant.

TTBS is a special case of anchor search with more sophisticated procedures for determining the anchors and candidate sets. During the process of finding the next state to expand, TTBS enumerates the highest priority states in the open list until a state fulfills a similarity criterion concerning the top state of the opposite open list. Hence, for each expansion, the set of states evaluated, among which the next state to expand is chosen, can be viewed as the candidate set, and each open list's top state represents the corresponding frontier's anchor.

In this chapter, we provided a detailed description of the anchor search framework for designing unbounded suboptimal Bi-HS algorithms. We listed various design choices for anchor and candidate selection policies offered by anchor search, introducing three distinct classes: brute-force, temporal, and randomized anchor search. Additionally, we discussed how some existing unbounded suboptimal Bi-HS algo-

| **Anchor Selection** | **Candidate Selection** | | |
|---|---|---|---|
| | **Temporal (T)** | **Randomized (R)** | **Brute-force (B)** |
| **Temporal (T)** | $\text{AS}_T^T$ | $\text{AS}_T^R$ | - |
| **Closest-to-the-opposite-anchor (A)** | $\text{AS}_A^T$ | $\text{AS}_A^R$ | - |
| **Fixed (F)** | - | - | BGBFS |
| **Hybrid(AF)** | $\text{AS}_{AF}^T$ | $\text{AS}_{AF}^R$ | - |

Table 3.1: Anchor search algorithms classified based on their anchor and candidate selection policies.

rithms, such as BGBFS, DNR, and TTBS, can be viewed as instances of anchor search. Table 3.1 classifies six proposed anchor search algorithms, alongside BGBFS, based on their anchor and candidate selection policies.

# Chapter 4

# Front-to-front Heuristics

The availability of a F2F heuristic is a crucial consideration for employing the anchor search framework, or any F2F Bi-HS algorithm effectively. Some domains naturally provide F2F heuristics (e.g., the octile distance for grid pathfinding or the Manhattan distance for the sliding tile puzzle). In cases where such heuristics are absent, we have found that embeddings, such as differential heuristics (DHs) [31] and FastMap [47, 48] can be used to get F2F heuristics estimates from existing F2E heuristics. We observe that embeddings can be applied in domains that do not fit in memory, even though they have traditionally been used in the literature primarily for explicit domains.

## 4.1 Metric Embeddings and F2F Heuristics

In mathematics, given a set of points $V$, a *metric* is defined as a function, denoted as $f : V \times V \to \mathbb{R}^+$, satisfying the following axioms:

1. $f(a, b) = 0 \Leftrightarrow a = b$
2. $f(a, b) \geq 0$
3. $f(a, b) = f(b, a)$
4. $f(a, b) + f(b, c) \geq f(a, c)$, known as the triangle inequality.

Then, pair $(V, f)$ is called a *metric space*. If all the above axioms hold except 1, i.e., there exists at least one pair of distinct points $(a, b)$ for which $f(a, b) = 0$,

the function $f$ is called a *semi-metric*. Metric spaces can be extended to *metric embeddings* where an injective function $\phi : V_1 \to V_2$ maps one metric space $(V_1, f_1)$ to another $(V_2, f_2)$. Although not widely used in the literature, we will adopt the term *semi-metric embedding* to refer to embedding of a metric space into a semi-metric space.

A class of metrics of interest in search problems includes *graph metrics*. In an arbitrary undirected graph $G(V, E)$, the pair $(V, d)$ forms a metric space. Here, the graph metric $d$, representing the shortest path costs between any pair of states in $V \times V$, is called the *shortest path metric*. If $d$ were known in advance, solving the shortest path problem would be trivial. To obtain $d$ for a given state space, one can employ A$^*$ search to compute the all-pairs shortest path data [49] for every pair of states. However, this approach is computationally demanding. While successful in polynomial domains, it is not scalable to implicit domains, such as exponential domains.

Metric and semi-metric embeddings have been widely used to embed complex graph metric spaces, with their computationally expensive shortest path metrics, into metric spaces with efficient metric functions estimating the shortest path cost between states [31, 42, 47]. In a state space $V$, a heuristic $h$ is obtained by embedding the metric space $(V, d)$ into a latent metric or semi-metric space $(V', h)$ using an embedding function $\phi : V \to V'$. Then, $h(\phi(a), \phi(b))$ serves as an estimate of $d(a, b)$. $h(\phi(a), \phi(b))$ is typically either less expensive than $d(a, b)$ to compute on-demand or is tractable to be fully stored in memory once and then used for solving many problem instances. The resultant heuristic, although typically not as informed as the respective shortest path metric, can guide search algorithms to expedite the search process. Many well-known heuristics applied to specific domains are obtained from metric embeddings as described above. For instance, the Manhattan distance in the sliding tile puzzle and the Euclidean and Octile distances in grid pathfinding are popular examples of metric embedding applications in heuristic search.

It is important to note that although the analogy between metric embeddings and heuristic design is prevalent in F2F heuristics, it does not apply to F2E heuristics. This is because a F2E heuristic does not estimate the distance between all pairs of states in $V \times V$, but only the pairs that include the start or goal state.

## 4.2 Differential Heuristic (DH)

A DH is a memory-based F2F heuristic, which in the literature has only been used in problems where the entire state space can be stored in memory [31]. The DH construction involves selecting a pivot $p$ and computing the precise distances $d(s, p)$ from all states $s \in V$ to $p$. This stored information enables the retrieval of heuristic values between any two states $a$ and $b$ with respect to $p$. For undirected graphs, this is formulated as $h(a, b) = |d(a, p) - d(b, p)|$. In scenarios with multiple pivots (heuristics), the resulting heuristic is the maximum among all the available heuristics: $h(a, b) = \max_{p \in pivots} |d(a, p) - d(b, p)|$. As the number of pivots increases, the DH gradually converges toward an all-pairs shortest-path heuristic [49], another memory-based heuristic that calculates and stores $d(a, b)$ for all pairs $(a, b) \in V \times V$.

We can demonstrate the semi-metric nature of DHs. Let's consider a DH, denoted as $h$, with $n$ pivots. It is straightforward to show that $h$ is consistently non-negative, yielding zero for identical states, and exhibiting symmetry. In fact, $h$ maps all states into an $n$-dimensional non-negative real coordinate space, where the triangle inequality holds for any arbitrary 3-tuple of points. The semi-metric property is due to the fact that two distinct states could have a heuristic value of zero.

Memory-based F2F heuristics such as DHs are not directly applicable to domains that do not fit in memory [31]. As a result, the study of implicit domains, lacking natural F2F heuristics, such as the 4-peg towers of Hanoi (TOH), has been mostly confined to solving problems with a standard goal [2]. We propose that the concept of DHs extends beyond domains that can fit in memory. In a large state space $V$, where it is impractical to find and store $d(a, p)$ for all states $a \in V$ and a chosen pivot

$p$, a pattern database (PDB) can be used to map $V$ to a more compact abstraction $\phi(V) = V'$. In this process, distances from $\phi(p) = p'$ to all states $s' \in \phi(V)$ are stored in a table. Consequently, when dealing with a state $a \in V$, the DH can efficiently retrieve $d(\phi(a), \phi(p))$ from the table, using it as an admissible estimate of $d(a, p)$. This estimate can then be used as a general F2F heuristic formulated as $h(a, b) = |d(\phi(a), \phi(p)) - d(\phi(b), \phi(p))|$.

Utilizing PDBs as DHs introduces the challenge of pivot placement [31, 50, 51]. The quality of a DH heavily relies on the pivots used for building the PDBs. Consider a given PDB, with $p'$ denoting its abstracted pivot. Let $a'$ and $b'$ constitute an arbitrary pair of abstracted states. Depending on the specific abstracted state $p'$, the heuristic function $h(a', b') = |d(a', p') - d(b', p')|$ can yield values ranging from zero to $d(a', b')$. The ideal scenario occurs when $h(a', b')$ equals $d(a', b')$, implying that either $a'$ lies on the shortest path between $b'$ and $p'$, or vice versa. Conversely, the worst-case scenario emerges when $d(a', p')$ equals $d(b', p')$, resulting in a heuristic value of zero. Based on our preliminary experiments, it turned out that a single large PDB for all problems with random start/goal states is less effective than smaller instance-specific PDBs. Thus, our experimental approach involves constructing two PDBs at runtime for each problem instance: one with the start state as the pivot and another with the goal state as the pivot. The following section provides an in-depth look at the practical limitations of instance-independent DHs through the use of a statistical model.

## 4.3 Insufficiency of Instance-independent DHs

Exponential domains typically have exponentially many states equidistant from the goal state. For instance, in the 4-peg towers of Hanoi (TOH) with 6 disks, over 20% of states are situated at a depth of 13 from the standard goal state. In our experiments involving TOH and the $4 \times 4$ sliding tile puzzle as two exponential domains, we found that using a PDB heuristic as a DH for F2F search resulted in a significant degradation of the heuristic quality (i.e., much smaller heuristic values). In other words, the

expected heuristic value of a PDB when being used as a DH to obtain heuristics for randomly sampled pairs of states is typically much lower than the expected heuristic value of the underlying PDB when used as a F2E heuristic (extracted from the PDB's histogram). For instance, when considering the TOH problem with 14 disks, we observed that a single-look-up PDB of size 7 built on the standard goal state has an expected heuristic value of 17.05, whereas when the same PDB used as a DH between arbitrary states it has an expected heuristic value of only 3.67.

This motivated the development of a simplified statistical model to predict the number of pivots needed to build a DH based on PDBs with the same expected heuristic value as a single PDB. In this model, we assume that all the possible PDBs built using arbitrary pivots chosen from $V$ have the same heuristic distribution. We refer to the PDB heuristic function as $h_{\text{PDB}} : V \to \mathbb{N}^0$ and to its corresponding differential heuristic function as $h_{\text{DH}} : V \to \mathbb{N}^0$. Note that a PDB heuristic's range is constrained solely to non-negative integer values. Let $H$ be the heuristic distribution of $h_{\text{PDB}}$, or the range distribution of the PDB heuristic function, and $X \sim H$ be a random variable representing the heuristic value obtained from $h_{\text{PDB}}$ for a state randomly sampled from $V$. Using $X$, we can express the heuristic value acquired from $h_{\text{DH}}$ for a pair of states randomly and independently selected as the random variable $Y = |X_1 - X_2|$. By denoting the probability density function (PDF) of $H$ as $f_X(x)$, we can derive $f_Y(y)$ as follows:

$$W = X_1 - X_2, Y = |W| \tag{4.1}$$

$$P(W = w) = f_W(w) = \sum_{m=-\infty}^{+\infty} f_X(m + d) f_X(m) \tag{4.2}$$

$$P(Y = y) = f_Y(y) = \begin{cases} f_W(0) & y = 0 \\ 2f_W(y) & y > 0 \\ 0 & y < 0 \end{cases} \tag{4.3}$$

The probability that the difference between two heuristic values $X_1 \sim H$ and

$X_2 \sim H$, associated with randomly and independently chosen states equals $d$, as given by Equation (4.2), is equivalent to computing the auto-correlation of the PDF of $H$. In other words, this equation determines the fraction of pairs of states for which the difference in heuristic values is equal to $d$, out of all possible pairs of states. This provides an intuitive explanation for Equation (4.2).

Since the auto-correlation of a function is symmetric about zero, $f_W(w)$ is also symmetric, that is, $f_W(w) = f_W(-w)$ for all values of $w$ in the support of $f_W$. Thus, for $w > 0$, the probability density function $f_{|W|}(w)$ can be expressed as $f_W(w) + f_W(-w)$, or equivalently, $2f_W(w)$. Moreover, as the absolute value function does not affect zero and is always non-negative, $f_{|W|}(0) = f_W(0)$ and $f_{|W|}(w) = 0$ for $w < 0$. This is how Equation (4.3) is derived as a piecewise function.

Thus far, we have derived the probability density function of $h_{\mathrm{DH}}$, which is constructed based on a single PDB. We now introduce a random variable, $Z^{(n)} = \max(Y_1, Y_2, ..., Y_n)$, which represents the differential heuristic value obtained from $n$ PDBs with randomly selected pivots. Equations (4.4) to (4.8) show how the PDF of $Z^{(n)}$ can be calculated using the cumulative distribution function (CDF) of $Y$, referred to as $F_Y(y)$.

$$F_{Z^{(n)}}(z) = P(Z^{(n)} \leq z) \tag{4.4}$$

$$= P(Y_1 \leq z, ..., Y_n \leq z) \tag{4.5}$$

$$= \prod_{i=1}^{n} P(Y_i \leq z) = F_Y(z)^n \tag{4.6}$$

Using the CDF of $Z^{(n)}$, the PDF of $Z^{(n)}$, $f_{Z^{(n)}}$ can be computed:

$$f_{Z^{(n)}}(z) = P(Z^{(n)} = z) \tag{4.7}$$

$$= P(Z^{(n)} \leq z) - P(Z^{(n)} \leq z - 1) \tag{4.8}$$

$$= F_Y(z)^n - F_Y(z-1)^n \tag{4.9}$$

After computing the probability density functions of $Y$ and $Z^{(n)}$ with various values

of $n$, we can derive their expected heuristics and then determine the minimum number of look-ups required to construct a DH that is as effective as a single goal-specific PDB.

We applied this theory to predict the potential usefulness of PDBs as DHs in three domains: the 4-peg Towers of Hanoi (TOH) with 14 disks, the 4-by-4 Sliding Tile Puzzle (STP) (also known as the 15-puzzle), and Rubik's Cube (RC). For TOH, we considered a PDB covering 7 consecutive disks. Note that the heuristic distribution does not depend on which consecutive disks are chosen. The expected heuristic value of this PDB is 17.05. According to Equation (4.9), to construct a DH with expected heuristic value higher than 17.05 would require taking the maximum of at least 625 DH embeddings.

For the 15-puzzle PDB, we used the upper half of the standard goal state (when the puzzle is solved), consisting of the blank tile plus 7 solid tiles. In our model the DH would requires 560,443 DH embeddings to obtain a higher expected heuristic value than the PDBs expected value of 38.91. Similarly, in Rubik's Cube, 11,668,052 DH embeddings would be needed for a DH to outperform its underlying PDB (with an expected heuristic value of 7.65) that captures 6 out of the 12 edges, in terms of the expected heuristic value.[1] Thus, theoretically, we cannot expect a F2F search (or anchor search in our case) to perform well when using instance-independent PDBs as DHs, given our assumptions. However, in practice, we deviate from these assumptions by constructing instance-dependent DHs. A deeper explanation will be provided in Chapter 5.

## 4.4 FastMap

Like DHs, FastMap [47, 48] is a metric embedding algorithm that can obtain metric embeddings for arbitrary undirected graphs in near-linear time. The resulting embedding enables calculating admissible and consistent F2F heuristics at runtime.

[1] Although in practice state symmetries can be used for arbitrary heuristic lookups with Rubik's Cube, we evaluate DHs in RC because the branching factor is large, meaning that most of the states are found in a very narrow range of depths [52, 53].

Initially, for a single dimension, the FastMap algorithm identifies an approximate furthest pair of states $a$ and $b$ in graph $G$ and, for all states $s \in V$, finds true distances $d(s,a)$ and $d(s,b)$. Then, an arbitrary state $s \in G$ will be embedded at location $\ell(s) = (d(a,s) + d(a,b) - d(s,b))/2$. Once all states are embedded, for an arbitrary pair of states $s_1$ and $s_2$, the resultant 1-dimensional FastMap heuristic is formulated as $h_{FM}(s_1, s_2) = |\ell(s_1) - \ell(s_2)|$. Note that in a single dimension, $|\ell(s_1) - \ell(s_2)|$ is equivalent to $||\ell(s_1) - \ell(s_2)||_1$, representing the $L^1$-norm. However, in higher dimensions, the $L^1$-norm notation is more suitable, as $\ell$ would yield vectors rather than single values.

This procedure can be repeated to improve the FastMap heuristic at the cost of extra memory. To do so, after constructing a 1-dimensional FastMap heuristic in the initial iteration on the original graph $G$, a residual graph $G'$ is created and employed for subsequent iterations. This residual graph shares the same vertices and edges as $G$, but it employs a distinct cost function for its edges. Specifically, the cost function of $G'$ during the $i$th iteration is defined as $c_i(u,v) = c_{i-1}(u,v) - |\ell_{i-1}(u) - \ell_{i-1}(v)|$, where $i$ is greater than 1, and $\ell_i$ signifies the embedding function obtained during the $i$th iteration. Since throughout each iteration the FastMap algorithm tries to capture the distances based on the residual graph, the heuristic obtained in different iterations can be aggregated in an additive manner without losing admissibility, i.e., $h_{FM}(s_1, s_2) = ||\ell(s_1) - \ell(s_2)||_1 = \sum_i |\ell_i(s_1) - \ell_i(s_2)|$ [48]. Note that, unlike DHs, FastMap heuristics do not converge towards an all-pairs shortest-path heuristic with an increasing number of iterations.

FastMap heuristics share the same limitation as DHs, namely, their infeasibility when the domain does not fit in memory. However, the same workaround proposed in the idea of using PDBs as DHs can be adapted to make FastMap applicable in implicit state spaces. A FastMap heuristic can be constructed using a compact PDB abstraction of the original state space. This time, $\phi(G)$ will be used instead of $G$, and the FastMap heuristic will be formulated as $h_{FM}(s_1, s_2) = ||\ell(\phi(s_1)) - \ell(\phi(s_2))||_1 =$

$\sum_i |\ell_i(\phi(s_1)) - \ell_i(\phi(s_2))|$. As the residual cost function in the $i$th iteration depends on the one from the previous iteration, it becomes difficult to present a statistical model for the expected heuristic value of an $n$-dimensional FastMap heuristic. Therefore, instead of providing such a theoretical analysis for FastMap, we choose to conduct an empirical analysis, which is detailed in Chapter 5.

# Chapter 5

# Experimental Results

We evaluate the performance of the anchor search algorithms in comparison to the baseline algorithms: GBFS, BGBFS, DNR with 100 consecutive expansions before retargeting, and two versions of TTBS with the alternating direction selection policy, using FIFO and LIFO as the tie-breaking mechanism, referred to as TTBS(F) and TTBS(L), respectively. The evaluation is done in terms of state expansions, time, and solution quality (length) on three problem domains: grid pathfinding, the $4 \times 4$ sliding tile puzzle (STP), and the 4-peg towers of Hanoi (TOH). Among $\text{AS}^T$ variants also, we only chose three specific instances: $\text{AS}_T^{T(10)}$, $\text{AS}_A^{T(10)}$, and $\text{AS}_{AF}^{T(10)}$, each of which showed a strong performance in at least one domain. Additionally, we included their randomized counterparts: $\text{AS}_T^{R(10)}$, $\text{AS}_A^{R(10)}$, and $\text{AS}_{AF}^{R(10)}$. The TOH and STP experiments were conducted using an Intel Gold 6148 Skylake CPU operating at 2.4GHz, with 180GB of available memory. Notably, we increased the memory limit to 700GB for a supplementary TOH experiment. Additionally, for grid pathfinding, we utilized an Intel E5-2683 v4 Broadwell CPU running at 2.1GHz, featuring 250GB of available memory. These computational resources were provided by Compute Canada. Before proceeding with the details, it is worth mentioning that the primary strength of TTBS when evaluated in planning was that it solves problems GBFS cannot, increasing overall coverage when both are used together [37].

## 5.1 Grid Pathfinding

We evaluated the algorithms using four map sets from the MovingAI pathfinding benchmarks [54]: Dragon Age: Origins (DAO), Starcraft 1 (SC1), Warcraft 3 (WC3), and mazes, with the octile distance as the heuristic. As the grid map state spaces are small, we used a pre-allocated table for storing the open/closed lists in GBFS, BGBFS and the anchor search variants, which is much faster than a general hash table.

The results, presented in Tables 5.1 and 5.2, underscore the robust performance of $\text{AS}^T_T$. It notably outperformed all five baseline algorithms in expansions for the DAO and mazes benchmarks, while being marginally outperformed by the leading algorithm, TTBS(L), in SC1 and WC3. Meanwhile, all $\text{AS}^T$ algorithms exhibited a remarkable improvement in execution time, showcasing an order of magnitude reduction compared to the baseline methods due to the constant time complexity of expansion. Importantly, the solution lengths achieved by $\text{AS}^T$ algorithms also remained on par with those of the baselines. For $\text{AS}^R$ variants, although they did not emerge as top performers in terms of expansions in this experiment, they exhibited significantly faster runtime compared to the baselines. This was due to the same reason as observed in $\text{AS}^T$ variants. Comparing $\text{AS}^T$ and $\text{AS}^R$ highlights the superiority of each $\text{AS}^T$ variant over its corresponding $\text{AS}^R$ variant in terms of expansions, runtime, and solution quality. Additionally, when comparing GBFS and its bidirectional counterpart, BGBFS, the results highlight the dominance of GBFS over BGBFS (except for the WC3 benchmark) in terms of expansion and runtime.

## 5.2 4-peg Towers of Hanoi (TOH)

We next evaluate the algorithms in TOH. In TOH, algorithms are typically run to the standard goal state [22]; we take up the more challenging problem of searching between random states. This means solution symmetry and large pre-computed

| Benchmarks | Top Baselines | | Temporal Anchor Search | | | Randomized Anchor Search | | |
|---|---|---|---|---|---|---|---|---|
| | **GBFS** | **TTBS(L)** | $\mathbf{AS}_{T}^{T(10)}$ | $\mathbf{AS}_{A}^{T(10)}$ | $\mathbf{AS}_{AF}^{T(10)}$ | $\mathbf{AS}_{T}^{R(10)}$ | $\mathbf{AS}_{A}^{R(10)}$ | $\mathbf{AS}_{AF}^{R(10)}$ |
| | | | | **Expansions** | | | | |
| **DAO** | 4,310 | 4,177 | **3,809** | 6,586 | 5,893 | 4,729 | 7,514 | 6,950 |
| **SC1** | 11,293 | **8,952** | 9,174 | 16,159 | 14,525 | 10,980 | 18,500 | 16,890 |
| **WC3** | 1,887 | **1,252** | 1,330 | 2,660 | 2,045 | 1,964 | 3,056 | 2,591 |
| **Mazes** | 45,694 | 43,496 | **41,509** | 52,827 | 50,325 | **41,317** | 60,093 | 56,986 |
| | | | | **Time (ms)** | | | | |
| **DAO** | 9.10 | 20.47 | **1.91** | 3.53 | 2.99 | 3.80 | 7.01 | 5.21 |
| **SC1** | 25.37 | 44.53 | **5.53** | 10.13 | 8.63 | 9.23 | 16.26 | 14.18 |
| **WC3** | 7.40 | 5.21 | **0.75** | 1.49 | 1.13 | 1.84 | 2.79 | 2.29 |
| **Mazes** | 40.90 | 210.01 | **21.31** | 30.33 | 26.48 | 32.28 | 48.61 | 44.93 |
| | | | | **Solution Length** | | | | |
| **DAO** | **473** | 581 | 499 | 481 | 489 | 517 | 490 | 495 |
| **SC1** | **690** | 933 | 746 | 706 | 731 | 871 | 763 | 780 |
| **WC3** | **387** | 470 | 412 | **391** | 416 | 513 | 444 | 444 |
| **Mazes** | 2,270 | 2,487 | 2,289 | **2,248** | 2,257 | 2,305 | 2,259 | 2,268 |

Table 5.1: Average expansions, time (milliseconds), path length, and their 95% confidence interval, in grid pathfinding (top baselines compared to the AS variants).

heuristics, which were necessary to scale optimal solutions to 30 disks [2], cannot be exploited. TOH lacks a natural F2F heuristic, and thus makes a good testbed for assessing the effectiveness of employing PDBs as DHs. To evaluate performance, especially with regard to larger instances, we conducted three distinct experiments for TOH. The initial experiment (Table 5.3) encompassed all studied algorithms, engaging them in solving small TOH instances with 10 disks. In the second experiment (Table 5.4), we narrowed our attention to the top-performing algorithms from the first experiment, namely $\text{AS}_{AF}^{T}$, BGBFS, and GBFS, as they were employed to solve larger TOH problems with 22 and 24 disks. The strong performance of $\text{AS}_{AF}^{T}$ in the second experiment motivated us to conduct a third experiment, aimed at solving larger TOH problems with 26, 28, 30, and 32 disks.

| Benchmarks | Top AS Variant | Baselines | | | | |
|---|---|---|---|---|---|---|
| | $AS_T^{T(10)}$ | GBFS | BGBFS | DNR | TTBS(L) | TTBS(F) |
| | | | **Expansions** | | | |
| **DAO** | **3,809** | 4,310 | 5,283 | 4,266 | 4,177 | 4,199 |
| **SC1** | 9,174 | 11,293 | 12,205 | 10,983 | **8,952** | **9,006** |
| **WC3** | 1,330 | 1,887 | 1,348 | 1,862 | **1,252** | **1,264** |
| **Mazes** | **41,509** | 45,694 | 49,327 | 47,160 | 43,496 | 43,520 |
| | | | **Time (ms)** | | | |
| **DAO** | **1.91** | 9.10 | 19.93 | 22.35 | 20.47 | 20.67 |
| **SC1** | **5.53** | 25.37 | 50.09 | 67.54 | 44.53 | 49.39 |
| **WC3** | **0.75** | 7.40 | 16.96 | 8.31 | 5.21 | 5.03 |
| **Mazes** | **21.31** | 40.90 | 61.38 | 340.80 | 210.01 | 205.67 |
| | | | **Solution Length** | | | |
| **DAO** | 499 | **473** | **477** | 486 | 581 | 576 |
| **SC1** | 746 | 690 | **684** | 729 | 933 | 917 |
| **WC3** | 412 | 387 | **381** | 404 | 470 | 463 |
| **Mazes** | 2,289 | 2,270 | 2,272 | **2,217** | 2,487 | 2,472 |

Table 5.2: Average expansions, time (milliseconds), path length, and their 95% confidence interval, in grid pathfinding (the top AS variant compared to the baselines).

In all experiments, we utilized DHs built upon PDBs using the start and goal states as the pivots. In the first experiment, the PDBs contained a single lookup capturing the five largest disks. In the second experiment, in addition to solving larger problems, we studied the effect of using different combinations of PDBs employed as DHs. More precisely, we conducted the second experiment in three settings:

1. **One strong lookup**: using a single lookup capturing the 12 largest disks, referred to as PDB(0-11),

2. **One strong + one weak lookup**: using PDB(0-11) in addition to a lookup

capturing the 4 largest remaining disks, referred to as PDB(12-15),

3. **Two strong lookups**: using PDB(0-11) as well as a lookup capturing all the disks not included in PDB(0-11), referred to as PDB(12-$\infty$).

In settings with two lookups, we combined the lookups in a weighted additive manner, multiplying PDB(0-11) by 100 to make it the primary guide for the search. The purpose of the second lookup (in the second and third settings) was to provide guidance once the algorithm solved the 12 largest disks. By magnifying PDB(0-11), we aimed to preserve its gradient (at the cost of losing admissibility) and only use the other lookup when PDB(0-11) does not provide guidance later during the search. While employing multiple additive lookups in a regular manner (equally weighted) reduces the work required for optimal planning, our preliminary experimental results demonstrated the opposite in unbounded suboptimal settings, consistent with previous findings [34]. We will refer to TOH with $n$ disks as TOH($n$).

**First TOH Experiment**

In the initial experiment (Table 5.3), $\text{AS}^T_{AF}$ and BGBFS, and GBFS emerged as standout performers, exhibiting significantly reduced expansion rates and improved runtimes. Notably, $\text{AS}^T_{AF}$ and BGBFS were the most performant algorithms in terms of expansions and runtime, respectively. It is worth highlighting that A$^*$ yielded an average of 209,867 expansions in the first experiment—roughly 300 times more than $\text{AS}^T_{AF}$—accompanied by a solution length of 30.

**Second TOH Experiment**

In the second experiment (Table 5.4), when utilizing single PDB lookups as DHs (the first setting), $\text{AS}^T_{AF}$ consistently outperformed both BGBFS and GBFS, with the performance gap widening as the problem size increased. More precisely, GBFS expanded approximately 3.3 and 5.9 times as many states as $\text{AS}^T_{AF}$ on average for the problem sizes of 22 and 24, respectively. Meanwhile, BGBFS expanded 2.5 and

| Metrics | Top AS Variants | | Baselines | | | | |
|---|---|---|---|---|---|---|---|
| | $\mathbf{AS}_{AF}^{T(10)}$ | $\mathbf{AS}_{AF}^{R(10)}$ | **GBFS** | **BGBFS** | **DNR** | **TTBS(L)** | **TTBS(F)** |
| **Expansions** | **707** | 809 | 1,150 | 903 | 7,462 | 17,098 | 12,596 |
| **Time (ms)** | 1.84 | 2.36 | 1.77 | **1.10** | 924.83 | 86.82 | 83.63 |
| **Length** | 483 | 219 | 801 | 604 | 2,427 | 1,943 | **75** |

(a) Top-AS variants vs. all baselines

| Metrics | Top Baselines | | Temporal AS | | | Randomized AS | | |
|---|---|---|---|---|---|---|---|---|
| | **GBFS** | **BGBFS** | $\mathbf{AS}_{T}^{T(10)}$ | $\mathbf{AS}_{A}^{T(10)}$ | $\mathbf{AS}_{AF}^{T(10)}$ | $\mathbf{AS}_{T}^{R(10)}$ | $\mathbf{AS}_{A}^{R(10)}$ | $\mathbf{AS}_{AF}^{R(10)}$ |
| **Expansions** | 1,150 | 903 | 5,579 | 7,491 | **707** | 15,117 | 10,929 | 809 |
| **Time (ms)** | 1.77 | **1.10** | 18.46 | 23.85 | 1.84 | 54.73 | 40.33 | 2.36 |
| **Length** | 801 | 604 | 1,224 | 844 | 483 | 450 | 348 | **219** |

(b) Top baselines vs. all AS variants

Table 5.3: TOH(10) results: average expansions, time, and solution length of 100 randomly selected problem instances.

3.5 times as many states as $\text{AS}_{AF}^{T}$ for the same problem sizes. Regarding average runtime, $\text{AS}_{AF}^{T}$ exhibited superiority, being 2 and 1.9 times faster than GBFS and BGBFS in TOH(22), and 3.6 and 3 times faster in TOH(24). $\text{AS}_{AF}^{T}$ also strongly outperformed GBFS and BGBFS in terms of solution length. Comparing the average expansions and solution length of $\text{AS}_{AF}^{T}$ in this experiment, it can be seen that the paths found contain more than 70% of the states expanded. Thus, $\text{AS}_{AF}^{T}$ tends to explore two paths greedily in both directions until they collide.

In the second setting, where strong and weak lookups were combined, all three algorithms exhibited a significant degradation in their expansions and runtime, while their solution lengths improved. Notably, GBFS emerged as the fastest algorithm in this setting, in contrast to the previous setting, and $\text{AS}_{AF}^{T}$ continued to be the top performer in terms of expansions and solution length.

In the third setting, employing two strong lookups, although $\text{AS}_{AF}^{T}$ still demonstrated fewer average expansions, the trend shifted in favor of GBFS and BGBFS in

| Metrics | 22 disks | | | 24 disks | | |
|---|---|---|---|---|---|---|
| | $\mathbf{AS}_{AF}^{T(10)}$ | **GBFS** | **BGBFS** | $\mathbf{AS}_{AF}^{T(10)}$ | **GBFS** | **BGBFS** |
| | | | **PDB(1-11)** | | | |
| **Expansions** | **413,828** | 1,349,162 | 1,020,929 | **2,805,473** | 16,686,368 | 9,921,693 |
| **Time (s)** | **3.01** | 5.98 | 5.84 | **20.80** | 74.71 | 63.17 |
| **Length** | **313,951** | 943,931 | 804,050 | **1,971,953** | 9,302,241 | 7,263,891 |
| | | | **PDB(1-11)×100 + PDB(12-15)** | | | |
| **Expansions** | **3,861,445** | 7,612,313 | 9,403,613 | **60,908,811** | 124,683,519 | 158,240,625 |
| **Time (s)** | 29.12 | **17.28** | 26.45 | 506.49 | **345.52** | 509.30 |
| **Length** | **123,742** | 558,635 | 540,627 | **712,199** | 8,053,873 | 8,021,651 |
| | | | **PDB(1-11)×100 + PDB(12-∞)** | | | |
| **Expansions** | **1,157,303** | 1,441,166 | 1,925,652 | **11,216,110** | 15,026,181 | 17,946,659 |
| **Time (s)** | 8.88 | **4.27** | 7.11 | 85.90 | **56.63** | 77.88 |
| **Length** | 11,811 | **1,568** | 1576 | 33,943 | **2,421** | 2397 |

Table 5.4: TOH(22) and TOH(24) results: average expansions, time (seconds), and solution length of 100 randomly selected problem instances.

| Metrics | PDB(12) - 180 GB | | PDB(15) - 700 GB | |
|---|---|---|---|---|
| | **26 disks** | **28 disks** | **30 disks** | **32 disks** |
| **Expansions** | 12,952,752 | 57,212,173 | 47,699,620 | 209,124,975 |
| **Time (s)** | 82.20 | 488.61 | 397.93 | 2,659.35 |
| **Length** | 9,673,562 | 42,792,775 | 36,077,524 | 162,213,243 |

Table 5.5: $\mathrm{AS}_{AF}^{T(10)}$ solving TOH problems with 26, 28, 30, and 32 disks.

terms of solution length. $AS_{AF}^{T}$ exhibited more than an order of magnitude longer average solution length than both GBFS and BGBFS in TOH(24). Additionally, note that both GBFS and BGBFS dominated $AS_{AF}^{T}$ in terms of runtime in this setting, in both TOH(22) and TOH(24).

It is worth noting that in the first setting with single lookups, once GBFS and BGBFS solve the 12 largest disks, the heuristic provides no guidance thereafter. These algorithms then perform a random walk blindly. However, in $AS_{AF}^{T}$, the backward frontier expands towards a moving anchor, always being guided during the search by the DH. If the remaining disks are also captured by another *spare* lookup (the setting with two strong lookups), the spare lookup continues guiding GBFS and BGBFS after solving the bottom disks. One can adjust this guidance by varying the number of extra disks captured by the spare lookup. Moreover, having an extra lookup results in stronger heuristics, enhancing solution quality at the cost of distorting the heuristic's gradient and leading to more expansions in greedy algorithms.

**Third TOH Experiment**

In this extended experiment (Table 5.5), we utilized a single lookup capturing the 12 largest disks (one strong PDB lookup) in TOH(26) and TOH(28). For TOH(30) and TOH(32), we employed larger PDBs with 15 disks and increased the available memory to 700GB. GBFS and BGBFS failed to solve 14 and one of the TOH(26) problem instances, respectively, and all the TOH(28) instances due to exceeding the available memory of 180GB. Similarly, in problems with a larger number of disks, GBFS could solve 65 out of 100 instances of TOH(30) but none in TOH(32). Since GBFS failed to solve a significant number of problem instances, we did not try BGBFS in TOH(30) and TOH(32). However, $AS_{AF}^{T(10)}$ successfully solved all the given problem instances with 26, 28, 30, and 32 disks, with its performance profile detailed in Table 5.5.

In TOH experiments, it can be observed that the expansion rate of anchor search algorithms is slower than that of GBFS. This is in contrast to the grid pathfinding

experiments, where anchor search demonstrated a significantly faster expansion rate. The slower expansion rate in TOH stems from the fact that TOH has a larger state data structure and, more importantly, more complex heuristics. The efficiency of heuristic computations has a greater impact on anchor search algorithms than on GBFS, as the former involves performing more heuristic evaluations.

It is also important to mention that the bidirectional nature of $\text{AS}_{AF}^{T}$ is of importance in solving large TOH instances according to its meet-in-the-middle characteristic. For instance, $\text{AS}_{AF}^{T}$ solved 75% of TOH(28) problem instances with paths meeting closer to the middle than the start/goal states. Moreover, when $\text{AS}_{AF}^{T(10)}$ only expands in the forward direction, it only solved 16 out of 100 TOH(28) problems, meaning the success comes from the bidirectional nature of the search. (The backward search searches towards the anchor and not the start, so it only makes sense as part of a bidirectional search.)

## 5.3 Sliding Tile Puzzle (STP)

Table 5.6 shows the average expansions, time, and solution length of 100 problems with $i^{\text{th}}$ and $i+10^{\text{th}}$ Korf's instances [55] as the start/goal states and the Manhattan distance (MD) as the heuristic. While DNR achieved the minimum average expansions in this experiment, the overlapping 95% confidence intervals of the average expansions for $\text{AS}_{A}^{T}$ $(1186 \pm 130)$ and DNR $(1003 \pm 123)$ highlighted the insignificant difference in their performance in terms of expansions. Moreover, $\text{AS}_{A}^{T}$ emerged as the top performer regarding runtime, while $\text{AS}_{A}^{R}$ achieved the highest solution quality on average. However, $\text{AS}_{T}^{T}$ and TTBS(L) performed poorly in STP, despite their strong performance in grid pathfinding. These algorithms share a common premise: expanding states based on their distance to recently generated states in the opposite frontier should lead to fewer expansions. However, the results indicate that such algorithms are not promising when the mentioned assumption does not hold.

We conducted an additional STP experiment employing PDBs as DHs, utilizing two

PDBs per problem instance: one associated with the start state and another with the goal state. Each PDB consists of 8 additive lookups, including 4 rows and 4 columns, yielding a stronger heuristic than MD, but one that is inadmissible. Interestingly, in contrast to TOH, GBFS demonstrated a notable advantage over $\text{AS}^{T}_{AF}$ (250.18 vs. 430.58 average expansions), indicating that the effectiveness of PDBs as DHs depends on factors such as the problem domain and the strength of the employed PDBs. While $\text{AS}^{T}_{AF}$ expanded fewer than twice the states of GBFS in this experiment, its average execution time was approximately 4.42 times longer than GBFS. This contrast in runtime highlights the greater impact of heuristic function call efficiency on anchor search, given its requirement to evaluate the entire candidate set for each expansion. By contrast, unidirectional and F2E algorithms have to evaluate just the successors.

| **Metrics** | **Top Baselines** | | **Temporal AS** | | | **Randomized AS** | | |
|---|---|---|---|---|---|---|---|---|
| | **GBFS** | **DNR** | $\textbf{AS}^{T(10)}_{T}$ | $\textbf{AS}^{T(10)}_{A}$ | $\textbf{AS}^{T(10)}_{AF}$ | $\textbf{AS}^{R(10)}_{T}$ | $\textbf{AS}^{R(10)}_{A}$ | $\textbf{AS}^{R(10)}_{AF}$ |
| **Expansions** | 1,945 | **1,003** | 92,907 | 1,186 | 1,449 | 45,764 | 1,882 | 2,526 |
| **Time (ms)** | 3.43 | 8.25 | 181.04 | **2.13** | 2.54 | 143.04 | 3.79 | 5.05 |
| **Length** | 463 | 361 | 12,907 | 380 | 460 | 344 | **141** | 161 |

(a) Top baselines vs. all AS variants

| **Metrics** | **Top AS Variants** | | **Baselines** | | | | |
|---|---|---|---|---|---|---|---|
| | $\textbf{AS}^{T(10)}_{A}$ | $\textbf{AS}^{T(10)}_{AF}$ | **GBFS** | **BGBFS** | **DNR** | **TTBS(L)** | **TTBS(F)** |
| **Expansions** | 1,186 | 1,449 | 1,945 | 2,317 | **1,003** | 56,475 | 2,156 |
| **Time (ms)** | **2.13** | 2.54 | 3.43 | 5.50 | 8.25 | 288.63 | 9.45 |
| **Length** | 380 | 460 | 463 | 311 | 361 | 49,084 | **269** |

(b) Top AS variants vs. all baselines

Table 5.6: STP results: average expansions, time, and solution length of 100 problems instances.

## 5.4 F2F Heuristics

We conducted two experiments to gain a deeper understanding of the effectiveness of PDBs as DHs and FastMap heuristics on the TOH problem. In the first experiment (Table 5.7), we assessed the efficacy of using FastMap heuristics as F2F heuristics. This experiment was performed on TOH(10) with the FastMap heuristic constructed on TOH(5) as the abstract state space, making the results comparable with Table 5.3. The second experiment (Table 5.8) focuses on comparing the performance of instance-specific PDBs against instance-independent PDBs on TOH(12), with each PDB containing a single lookup capturing 6 consecutive disks.

Table 5.7, displays the average expansions performed by various algorithms, including anchor search variants like $\text{AS}^T_{AF}$ and $\text{AS}^R_{AF}$, serving as robust representatives of $\text{AS}^T$ and $\text{AS}^R$, along with the baselines. These algorithms were evaluated on 100 random TOH(10) problem instances, employing $d$-dimensional FastMap heuristics. The value of $d$ spans from 2 to 6, with a DH as the last dimension [48]. While the relative performance of the algorithms align with the trends observed in solving TOH using PDBs as DHs, it is notable that the overall effectiveness of FastMap is relatively lower than that of PDBs as DHs (Table 5.3).

Table 5.8 underscores the enhanced performance achieved by $\text{AS}^T_T$ and $\text{AS}^T_{AF}$, two anchor search algorithms employing different anchor selection policies, when *instance-specific* heuristics are incorporated. This improvement is evident in comparison to scenarios where only *instance-independent* PDBs with randomly selected pivots are used. We evaluated these algorithms in seven different settings: The first setting, denoted as 2I, indicates that the algorithm is provided with PDBs constructed solely on the start and goal states (two instance-specific PDBs), without including any PDBs with randomly selected pivots (no instance-independent PDB). In settings 10I, 20I, and 40I, the two instance-specific heuristics are bolstered by the inclusion of 8, 18, and 38 instance-independent PDBs with randomly selected pivots, respectively.

Finally, in settings 10S, 20S, 40S, the algorithms are exclusively provided with 10, 20, and 40 fully random instance-independent PDBs, in turn.

When comparing the performance of algorithms in settings with an equal number of total PDBs (e.g., 10I and 10S), it becomes apparent that including instance-specific PDBs significantly enhances performance in terms of expansions. Note that, this improvement comes at the expense of slower heuristic evaluations as the number of PDBs increases.

A specific comparison between $\text{AS}_{T}^{T(10)}$ in setting 2S and the remaining settings reveals the crucial role played by instance-independent PDBs in reducing the number of expansions. Conversely, for $\text{AS}_{AF}^{T(10)}$, the inclusion of instance-independent PDBs has a negligible impact. This suggests that interleaving F2F and F2E searches in opposite directions eliminates the need for instance-independent PDBs in $\text{AS}_{AF}^{T}$, leading to more efficient heuristic evaluations and faster runtime.

| **Dimensions (FM+DH)** | **Top AS Variants** | | **Baselines** | | | | |
|---|---|---|---|---|---|---|---|
| | $\textbf{AS}_{AF}^{T(10)}$ | $\textbf{AS}_{AF}^{R(10)}$ | **GBFS** | **BGBFS** | **DNR** | **TTBS(L)** | **TTBS(F)** |
| **2** | **18,520** | 26,892 | 44,926 | 18,575 | 21,277 | 40,874 | 51,764 |
| **3** | **12,824** | 20,051 | 28,128 | 15,251 | 18,264 | 30,343 | 36,667 |
| **4** | **11,792** | 18,376 | 27,758 | 15,216 | 16,329 | 33,549 | 35,552 |
| **5** | **13,314** | 17,599 | 28,523 | 15,715 | 14,319 | 26,125 | 31,541 |
| **6** | **13,128** | 17,139 | 25,576 | 13,957 | 15,388 | 27,193 | 31,637 |

Table 5.7: Average expansions in TOH(10) using FastMap heuristics.

## 5.5 Summary of Experiments

Overall, our results show that GBFS is a robust algorithm, and it has better performance than TTBS and DNR, as shown previously [37]. However, given the anchor search framework, $\text{AS}_{AF}^{T}$ is not only robust, but also consistently outperforms GBFS. With regard to anchor search variants, we observe that when using PDBs as DHs in

| Algorithms | The number of PDBs | | | | | | |
|---|---|---|---|---|---|---|---|
| | **2S** | **10S** | **20S** | **40S** | **10I** | **20I** | **40I** |
| $\mathbf{AS}_{T}^{T}$ | 51,482 | 7,179 | 3,639 | 2,719 | 27,369 | 11,326 | 7,338 |
| $\mathbf{AS}_{AF}^{T}$ | 2,073 | 1,985 | 1,982 | 1,899 | 39,497 | 18,944 | 9,291 |

Table 5.8: Comparison of average expansions with instance-specific and instance-independent PDBs as DHs in TOH(12). $n$I denotes $n$ instance-independent PDBs, while $n$S denotes $n-2$ instance-independent PDBs along with two instance-specific (start and goal) PDBs.

a pure F2F search, both frontiers will experience the same local minima mutually. When the search is diversified by adopting different heuristics in each direction (F2F and F2E), we see the best performance. This aligns with past work on the value of diversifying search [56], and suggests why $\text{AS}_{AF}^{T}$ is the most robust variant explored.

# Chapter 6

# Conclusions, Recommendations, & Future Work

This thesis proposed the anchor search framework that facilitates designing F2F suboptimal Bi-HS algorithms. We evaluated the potential of this framework through six specific anchor search algorithms in three domains of grid pathfinding, 4-peg towers of Hanoi (TOH), and the $4 \times 4$ sliding tile puzzle (STP), achieving promising performance against the baselines.

In grid pathfinding, $\text{AS}_T^T$ emerged as the top performer in expansions and runtime while maintaining comparable solution quality to the best baselines. In TOH, $\text{AS}_{AF}^T$ not only outperformed other algorithms in small problems but also showed synergy with PDBs when used as DHs, allowing it to solve very large problems, such as TOH(30) and TOH(32). The promising performance of $\text{AS}_{AF}^T$ extended beyond the TOH domain, as it also performed well in STP and grid pathfinding, showcasing overall robust performance. Finally, in STP, $\text{AS}_A^T$ was the second-best algorithm studied regarding expansions following DNR, but it stood out as the fastest in terms of runtime within this domain.

Despite the potential of anchor search in designing more complex algorithms, we focused on developing a range of simple variants to be able to better grasp and explain the proposed algorithms' behaviors. There are numerous design choices whose analyses were beyond the scope of this thesis. Although we only considered utilizing

a single anchor in this thesis, the anchor search framework can be more generalized by maintaining a set of anchors per frontier. In addition, while we provided insight into the potential of incorporating different anchor selection policies in the hybrid variants, namely $\text{AS}^T_{AF}$ and $\text{AS}^R_{AF}$, one can apply the same idea to candidate selection policy as well by adopting different candidate selection policies in opposite directions of the search. Moreover, while not discussed in this thesis, one can switch dynamically among a set of anchor and candidate selection policies in a single direction during the search, e.g., alternating between temporal and fixed-to-the-origin anchor selection policies in one frontier instead of employing them in opposite directions.

We also proposed the idea of using PDBs as DHs, which makes F2F search algorithms applicable in domains without natural F2F heuristics. Empirical analyses highlighted the significant promise of PDBs as DHs in solving TOH problems of various sizes, while STP results revealed its limitations when powered by exceptionally strong PDBs. It is worth noting that most of the heuristics used in this thesis are admissible, a property that is neither required nor helpful for unbounded suboptimal search [34]. Future work will consider other heuristic approaches that might have more potential in unbounded suboptimal search by focusing on relevant heuristic properties, such as a heuristic's capacity to rank pairs of states in the same order as their actual cost-based order [34, 35]. Furthermore, the effectiveness of rebuilding PDBs on-demand during the search to be used as DHs can be examined to solve even larger problems. Additionally, a greedy search can be employed instead of a breadth-first search to construct instance-specific PDBs as DHs, covering only the necessary portion of the abstract state space for solving the given problem instance. Then, this approach can be employed recursively in a hierarchical manner [57] to solve very large problems.

# Bibliography


[1] R. E. Korf and L. A. Taylor, "Finding optimal solutions to the twenty-four puzzle," in *Proceedings of the national conference on artificial intelligence*, 1996, pp. 1202–1207.

[2] R. E. Korf and A. Felner, "Recent progress in heuristic search: A case study of the four-peg towers of hanoi problem.," in *International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 20, 2007, 2324–2329.

[3] R. E. Korf, "Finding optimal solutions to rubik's cube using pattern databases," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 1997, pp. 700–705.

[4] Y. Björnsson and K. Halldórsson, "Improved heuristics for optimal pathfinding on game maps," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 2, 2006, pp. 9–14.

[5] R. Stern *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, 2019, pp. 151–158.

[6] E. Burns, W. Ruml, and M. B. Do, "Heuristic search when time matters," *Journal of Artificial Intelligence Research*, vol. 47, pp. 697–740, 2013.

[7] J. Thayer and W. Ruml, "Anytime heuristic search: Frameworks and algorithms," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 1, 2010, pp. 121–128.

[8] P. Haslum, A. Botea, M. Helmert, B. Bonet, S. Koenig, *et al.*, "Domain-independent construction of pattern database heuristics for cost-optimal planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 7, 2007, pp. 1007–1012.

[9] H. Kaindl and G. Kainz, "Bidirectional heuristic search reconsidered," *Journal of Artificial Intelligence Research*, vol. 7, pp. 283–317, 1997.

[10] L. Sint and D. De Champeaux, "An improved bidirectional heuristic search algorithm," *Journal of the ACM (JACM)*, vol. 24, no. 2, pp. 177–191, 1977.

[11] T. A. J. Nicholson, "Finding the shortest route between two points in a network," *The computer journal*, vol. 9, no. 3, pp. 275–280, 1966.

[12] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, 269–271, 1959.

[13] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[14] R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of A*," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 505–536, 1985.

[15] I. Pohl, "Bi-directional search," *Machine intelligence*, vol. 6, pp. 127–140, 1971.

[16] N. J. Nilsson, *Principles of artificial intelligence*. Springer Science & Business Media, 1982.

[17] J. Barker and R. Korf, "Limitations of front-to-end bidirectional heuristic search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, 2015, pp. 1086–1092.

[18] R. Holte, A. Felner, G. Sharon, and N. Sturtevant, "Bidirectional search that is guaranteed to meet in the middle," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016, pp. 3411–3417.

[19] E. Shaham, A. Felner, J. Chen, and N. Sturtevant, "The minimal set of states that must be expanded in a front-to-end bidirectional search," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 8, 2017, pp. 82–90.

[20] J. Eckerle, J. Chen, N. Sturtevant, S. Zilles, and R. Holte, "Sufficient conditions for node expansion in bidirectional heuristic search," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2017, pp. 79–87. [Online]. Available: http://www.cs.ualberta.ca/~nathanst/papers/eckerle17sufficient.pdf.

[21] N. R. Sturtevant, S. Shperberg, A. Felner, and J. Chen, "Predicting the effectiveness of bidirectional heuristic search," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2020, pp. 281–290. [Online]. Available: https://www.cs.ualberta.ca/~nathanst/papers/sturtevant2020unibidi.pdf.

[22] J. Chen, R. C. Holte, S. Zilles, and N. R. Sturtevant, "Front-to-end bidirectional heuristic search with near-optimal node expansions," *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017. [Online]. Available: http://www.cs.ualberta.ca/~nathanst/papers/chen2017nbs.pdf.

[23] S. Shperberg, A. Felner, N. R. Sturtevant, A. Hayoun, and E. S. Shimony, "Enriching non-parametric bidirectional search algorithms," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 2379–2386. [Online]. Available: http://www.cs.ualberta.ca/~nathanst/papers/DVCBS.pdf.

[24] J. Chen and N. R. Sturtevant, "Conditions for avoiding node re-expansions in bounded suboptimal search," *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019. [Online]. Available: https://webdocs.cs.ualberta.ca/~nathanst/papers/chen2019conditions.pdf.

[25] M. Likhachev and A. Stentz, "R* search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2008, pp. 344–350.

[26] Y. Björnsson, V. Bulitko, and N. R. Sturtevant, "TBA*: Time-bounded A*.," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2009, pp. 431–436.

[27] C. Wilt, J. Thayer, and W. Ruml, "A comparison of greedy search algorithms," in *Proceedings of the International Symposium on Combinatorial Search*, 2010, pp. 129–136.

[28] M. Heusner, T. Keller, and M. Helmert, "Understanding the search behaviour of greedy best-first search," in *Proceedings of the International Symposium on Combinatorial Search*, 2017, pp. 47–55.

[29] G. Politowski and I. Pohl, "D-node retargeting in bidirectional heuristic search.," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 1984, pp. 274–277.

[30] J. C. Culberson and J. Schaeffer, "Pattern databases," *Computational Intelligence*, vol. 14, no. 3, pp. 318–334, 1998.

[31] N. R. Sturtevant, A. Felner, M. Barer, J. Schaeffer, and N. Burch, "Memory-based heuristics for explicit state spaces," *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 609–614, 2009. [Online]. Available: http://www.cs.ualberta.ca/~nathanst/papers/TDH.pdf.

[32] E. Shaham, A. Felner, N. R. Sturtevant, and J. S. Rosenschein, "Minimizing node expansions in bidirectional search with consistent heuristics," in *Proceedings of the International Symposium on Combinatorial Search*, 2018, pp. 81–89. [Online]. Available: http://www.cs.ualberta.ca/~nathanst/papers/shaham18consistent.pdf.

[33] V. Alcázar, "The consistent case in bidirectional search and a bucket-to-bucket algorithm as a middle ground between front-to-end and front-to-front," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2021, pp. 7–15.

[34] C. Wilt and W. Ruml, "Effective heuristics for suboptimal best-first search," *Journal of Artificial Intelligence Research*, vol. 57, pp. 273–306, 2016.

[35] C. Wilt and W. Ruml, "Building a heuristic for greedy search," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 6, 2015, pp. 131–140.

[36] L. Siag, S. Shperberg, A. Felner, and N. R. Sturtevant, "Comparing front-to-front and front-to-end heuristics in bidirectional search," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 16, 2023, pp. 158–162.

[37] R. Kuroiwa and A. Fukunaga, "Front-to-front heuristic search for satisficing classical planning," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 4098–4105.

[38] L. E. Mayer and K. D. Krebsbach, "Front-to-front bidirectional best-first search reconsidered," in *The Thirty-Second International Flairs Conference*, 2019.

[39] F. Islam, V. Narayanan, and M. Likhachev, "A*-connect: Bounded suboptimal bidirectional heuristic search," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 2752–2758.

[40] D. Atzmon, S. S. Shperberg, N. Sabah, A. Felner, and N. R. Sturtevant, "W-restrained bidirectional bounded-suboptimal heuristic search," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 33, 2023, pp. 26–30.

[41] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artificial intelligence*, vol. 1, no. 3-4, pp. 193–204, 1970.

[42] J. E. Doran and D. Michie, "Experiments with the graph traverser program," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 294, no. 1437, pp. 235–259, 1966.

[43] F. Xie, M. Müller, and R. Holte, "Adding local exploration to greedy best-first search in satisficing planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, 2014.

[44] F. Xie, M. Müller, and R. Holte, "Jasper: The art of exploration in greedy best first search," *The Eighth International Planning Competition (IPC-2014)*, pp. 39–42, 2014.

[45] R. A. Valenzano, N. R. Sturtevant, J. Schaeffer, K. Buro, and A. Kishimoto, "Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms," in *Proceedings of International Conference on Automated Planning and Scheduling*, 2010, pp. 177–184. [Online]. Available: http://www.cs.ualberta.ca/~nathanst/papers/dovetailing.pdf.

[46] X. Sun, W. Yeoh, P.-A. Chen, and S. Koenig, "Simple optimization techniques for A*-based search," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 2009, pp. 931–936.

[47] L. Cohen, T. Uras, S. Jahangiri, A. Arunasalam, S. Koenig, and T. K. S. Kumar, "The fastmap algorithm for shortest path computations," in *International Joint Conference on Artificial Intelligence (IJCAI)*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2018, pp. 1427–1433. DOI: 10.24963/ijcai.2018/198. [Online]. Available: https://doi.org/10.24963/ijcai.2018/198.

[48] R. Mashayekhi, D. Atzmon, and N. R. Sturtevant, "Analyzing and improving the use of the fastmap embedding in pathfinding tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 12 473–12 481.

[49] A. Botea, "Ultra-fast optimal pathfinding without runtime search," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 7, 2011, pp. 122–127.

[50] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory.," in *Symposium on Discrete Algorithms (SODA)*, vol. 5, 2005, pp. 156–165.

[51] L. H. Levi, S. Franco, M. Abisrror, M. Barley, S. Zilles, and R. Holte, "Heuristic subset selection in classical planning," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2016, pp. 3185–3195.

[52] R. E. Korf, "Minimizing disk I/O in two-bit breadth-first search.," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2008, pp. 317–324.

[53] N. Sturtevant, A. Felner, and M. Helmert, "Exploiting the rubik's cube 12-edge pdb by combining partial pattern databases and bloom filters," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 5, 2014, pp. 175–183.

[54] N. R. Sturtevant, "Benchmarks for grid-based pathfinding," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144–148, 2012.

[55] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search," *Artificial intelligence*, vol. 27, no. 1, pp. 97–109, 1985.

[56] T. Imai and A. Kishimoto, "A novel technique for avoiding plateaus of greedy best-first search in satisficing planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, 2011, pp. 985–991.

[57] R. C. Holte, J. Grajkowski, and B. Tanner, "Hierarchical heuristic search revisited," in *International Symposium on Abstraction, Reformulation, and Approximation*, Springer, 2005, pp. 121–133.