Anchor Search: A Unified Framework for Suboptimal Bidirectional Search

Sepehr Lavasani[1], Lior Siag[2], Shahaf S. Shperberg[2], Ariel Felner[2], Nathan R. Sturtevant[1, 3]

[1] University of Alberta
[2] Ben-Gurion University of the Negev
[3]Alberta Machine Intelligence Institute (Amii)
sepehr3@ualberta.ca, siagl@post.bgu.ac.il, {shperbsh, felner}@bgu.ac.il, nathanst@ualberta.ca

Abstract

In recent years the understanding of optimal bidirectional heuristic search (BiHS) has progressed significantly. Yet, Bi-HS is relatively unexplored in unbounded suboptimal search. Front-to-end (F2E) and front-to-front (F2F) bidirectional search have been used in optimal algorithms, but adapting them for unbounded suboptimal search remains an open challenge. We introduce a framework for suboptimal BiHS, called *anchor search*, and use it to derive a parameterized family of algorithms. Because our new algorithms need F2F heuristic evaluations, we propose using pattern databases (PDBs) as differential heuristics (DHs) to construct F2F heuristics. Our experiments evaluate three anchor search instances across diverse domains, outperforming existing methods, particularly as the search scales.

Code — https://github.com/nathansttt/hog2/tree/PDB-refactor/papers/AnchorSearch

Introduction

In optimal heuristic search, the objective is to discover the path of least cost connecting two vertices, denoted as *start* and *goal*, within a given graph. *Unidirectional heuristic search* (UHS) algorithms search for a path by exploration from *start* towards the *goal* using the guidance of a heuristic function. An alternative approach to UHS is *Bidirectional heuristic search* (BiHS), which involves simultaneous exploration from both the *start* (forward) and *goal* (backward) directions until the two frontiers meet.

In numerous contexts, finding optimal solutions is impractical due to computational costs. *Suboptimal search* sacrifices solution quality in exchange for expedited execution times, and can be *bounded*, returning a solution within a fixed bound of optimal (Valenzano et al. 2013), or *unbounded*, returning any solution. In both of these cases, the search terminates when the first suitable solution is found, instead of continuing to look for better solutions, something done by anytime search (Hansen and Zhou 2007).

This work addresses the problem of unbounded suboptimal Bi-HS. In the past, optimal and suboptimal Uni-HS have been studied comprehensively (Dechter and Pearl 1985; Likhachev and Stentz 2008; Björnsson, Bulitko, and Sturtevant 2009; Wilt, Thayer, and Ruml 2010; Heusner, Keller, and Helmert 2017). Moreover, recent years have witnessed a considerable surge in attention towards optimal Bi-HS, resulting in not only refined theoretical understanding (Eckerle et al. 2017; Shaham et al. 2018; Chen and Sturtevant 2019; Shperberg et al. 2019; Sturtevant et al. 2020) but also the development of efficient algorithms (Holte et al. 2016; Chen et al. 2017; Shperberg et al. 2019; Alcázar 2021). Even though early work in bidirectional search studied suboptimal approaches (Sint and De Champeaux 1977), unbounded suboptimal Bi-HS algorithms remain relatively unexplored.

Bi-HS algorithms can be divided into two categories: front-to-end (F2E) (Kaindl and Kainz 1997) and front-to-front (F2F) algorithms (Sint and De Champeaux 1977). In F2E Bi-HS, heuristics estimate distances from states to either the *start* or *goal*, while in F2F Bi-HS, heuristics estimate the distances between any pair of states.

This paper introduces the *anchor search* framework, for unbounded suboptimal Bi-HS. Existing unbounded suboptimal Bi-HS algorithms fit loosely into this framework, and novel instances of anchor search arise from the algorithmic choices in the framework. This paper also addresses a previously overlooked question—the derivation of F2F heuristics, especially in domains like the Towers of Hanoi (TOH) where strong F2F heuristics are not readily available. Within this context, the paper explores how pattern databases (PDBs) (Culberson and Schaeffer 1998) can be used as differential heuristics (DHs) (Sturtevant et al. 2009), providing the F2F heuristics for anchor search in particular, and any F2F bidirectional search in general.

Experimental results across domains show that the new anchor search algorithms exhibit promising performance, surpassing existing unbounded suboptimal algorithms. Furthermore, our experiments in TOH show that using PDBs as DHs are effective F2F heuristics.

Background and Related Work

This paper addresses the challenge of finding paths of any length between two states (vertices) within a graph. The inputs consist 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. The out-

put is a path $\pi = v_0, ..., v_n$ with a cost of $\sum_{i=0}^{n-1} c(v_i, v_{i+1})$, where $v_i, v_{i+1} \in E$.

We assume that algorithms are expansion-based (Dechter and Pearl 1985). In each step a single state is selected and expanded. For each state s, $g(s)$ represents the cost of the shortest path found thus far between s and its corresponding frontier's origin (start/goal). $d(a, b)$ represents the cost of the optimal path between a and b.

There are two suboptimal search settings. **(1)** *bounded suboptimal search* (BSS) is defined relatively to a cost bound B. In BSS the task is to find a path with cost $\leq B$. **(2)** *unbounded suboptimal search* (UBS) where a possibly suboptimal path has to be found but not bound is given on its cost.

Bidirectional heuristic search algorithms (BiHS) simultaneously search forward from the start state and backwards from the goal state until the two search frontiers meet. Subscripts D and $\bar{D}$ are used to differentiate the current and opposite directions of the search. States and nodes are used interchangeably in this paper. Two types of heuristic are usually considered for BiHS: *front-to-end* (F2E) heuristics (Kaindl and Kainz 1997) which estimate the distance between any state and the $start$ or $goal$, and *Front-to-front* (F2F) heuristics (de Champeaux and Sint 1977) estimate the distance between any two nodes in the search space. F2F heuristics are more accurate than F2E heuristics, but working with them requires excessive computational overhead.

Optimal and BSS BiHS Algorithms

Two recent papers have shown the potential effectiveness of BSS Bi-HS search over Uni-HS. A*-connect (Islam, Narayanan, and Likhachev 2016) is a BSS (Bi-HS) algorithm that outperformed state-of-the-art unidirectional algorithms in motion planning. A*-connect utilizes the notion of pivots. Pivots are a set of representative states that encapsulate the search frontier. Atzmon et al. (2023) studied how the ideas behind Weighted A* (Pohl 1970) can be applied to the MM algorithm (Holte et al. 2016), resulting in Weighted MM (WMM). These algorithms are for the BSS setting and are not directly comparable to UBS algorithms like anchor search. Similarly, optimal algorithms (e.g., A* (Hart, Nilsson, and Raphael 1968), SFBDS (Felner et al. 2010), NBS (Chen et al. 2017), DVCBS (Shperberg et al. 2019), MM (Holte et al. 2016), etc.) are not suitable baselines.

Unbounded Suboptimal Baselines

This work studies the problem of *unbounded suboptimal* search (UBS), a satisficing search that terminates as soon as a path from the start to the goal is found. We consider the following UBS algorithms as baselines:

• **Greedy Best-First Search (GBFS)**, 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.[1] Given its simplicity and wide use in planning, GBFS is a clear choice for baseline comparisons.

• **BGBFS**, is a bidirectional version of GBFS. 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 (start or goal) 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.

• **DNR** The most informative approach to utilizing F2F heuristic involves computing heuristic estimations for pairs of states in opposite frontiers (Sint and De Champeaux 1977). Because, this approach is computationally demanding, Politowski and Pohl (1984) introduced the concept of a d-node: a dynamic representative state in each frontier, which serves as a temporary goal for the opposite frontier. d-node retargeting (DNR), is a UBS Bi-HS algorithm that uses d-nodes. In DNR, each frontier performs n consecutive expansions prioritized by the distance 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 $f = (1-w)g + wh$ 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.

• **TTBS**. Kuroiwa and Fukunaga (2021) proposed top-to-top bidirectional search (TTBS) as a modification of DNR. 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 of each open list as the d-node for that specific frontier. As TTBS is a complex algorithm we refer the reader to the complete description (Kuroiwa and Fukunaga 2021).

Anchor Search

F2F heuristics can be more informed than F2E heuristics, since they provide estimates between any pair of states. However, the quadratic complexity of assessing all pairs of frontier states required by optimal search (Eckerle et al. 2017) makes full F2F evaluations expensive (Siag et al. 2023). F2F BiHS algorithms can reduce this overhead by finding suboptimal solutions. With this in mind, we introduce the *anchor search* framework, which unifies multiple suboptimal F2F algorithms into a single framework.

The anchor search framework uses two main concepts: an *anchor* and *candidates*, illustrated in Figure 1. An *anchor* is a representative state associated with each frontier, treated as a provisional goal by the opposite frontier. In DNR, a d-node is the state with the largest g-cost; an anchor can be any state used as a target. As detailed in Algorithm 1, 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

[1] The policy for how GBFS should tie-break in case of equal h-cost is not specified as part of the algorithm. Our implementation prefers states with larger g-cost.

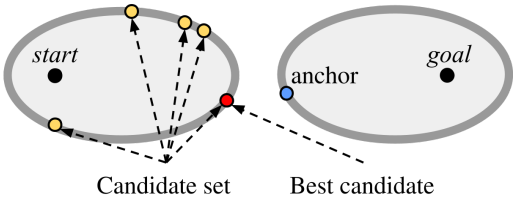

Figure 1: Anchor search illustration; the blue dot represents the backward frontier's anchor, the red dot shows the next state to expand, and the yellow dots are other candidates.

Algorithm 1: Anchor Search

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

1: **while** $Open_D$ is not empty **do**
2: $C \leftarrow$ GETCANDIDATES($Open_D$)
3: $next \leftarrow \underset{n \in C}{\text{argmin}}\ h(n, anchor_{\bar{D}})$
4: remove $next$ from $Open_D$
5: add $next$ to $Closed_D$
6: **for** $s \in$ SUCCESSORS($G, next$) **do**
7: **if** $s \notin Closed_D$ **then**
8: add s to $Open_D$
9: $parent(s) \leftarrow next$
10: **else if** $g(s) > g(next) + c(next, s)$ **then**
11: $g(s) \leftarrow g(next) + c(next, s)$
12: $parent(s) \leftarrow next$
13: **end if**
14: **if** $s \in Open_{\bar{D}}$ **then**
15: **return** EXTRACTPATH(s,$start$, $goal$)
16: **end if**
17: **end for**
18: $anchor_D \leftarrow$ GETANCHOR($Open_D, Closed_D$)
19: $D \leftarrow$ GETDIRECTION()
20: **end while**
21: **return** null

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 Algorithm 1, the GETANCHOR, GETCANDIDATES, and GETDIRECTION procedures correspond to the anchor selection, candidate selection, and direction selection policies that must be provided for any instance of anchor search.

Anchor Selection

The anchor selection policy governs the process of selecting the anchor from all the nodes within the frontier during each iteration. No single dominant 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). 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 $h(s, goal)$.
• **Closest to the opposite anchor:** The anchor is updated to the most recently 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:** The anchor is $start$/$goal$ and never changes.

The anchor selection policy can also include the frequency of anchor updates, either explicitly or implicitly. This flexibility facilitates the design of both F2E and F2F Bi-HS algorithms. If the anchor is fixed, the resulting algorithm aligns with F2E search. In scenarios where the anchor can vary, the algorithm aligns with F2F search. One possibility not evaluated here is the potential for multiple anchors; this is left for future work.

Candidate Selection

Anchor search assembles a set of candidates from which the next state for expansion is selected. It prioritizes states according to their h-cost to the opposite anchor. In contrast to optimal search algorithms, it is not necessary for anchor search to consider all states in open. The subset of the open list considered is defined by the candidate selection policy. The candidate-selection policies we consider are:
• **Brute-force:** Candidates are all states on the open list.
• **Temporal:** Candidates are selected from the k most recently generated states on the open list.
• **Random:** Candidates are a subset of states selected randomly from the open list.
• **Random+Temporal:** Candidates include the best state from *temporal* with the remaining candidates from *random*.

Temporal candidate selection resembles beam search (Bisiani 1992); however, unlike beam search, anchor search does not prune the open list. We do not compare directly to beam search, although we will report the results of GBFS with a temporal expansion policy.

Direction Selection

The direction selection policy determines the expansion direction in each iteration of the algorithm. An *alternating* policy toggles the expansion direction after every iteration. The k-alternating policy switches the expansion direction every k iterations. This direction change can also be contingent on conditions such as the maximum or minimum h-cost within a frontier. It is possible to craft Uni-HS algorithms anchor search by maintaining the same expansion direction throughout the search. All novel anchor search algorithms in this paper adopt the alternating policy.

Completeness of Anchor Search

Anchor search expands the best state among the candidates with respect to the opposite anchor, never re-expands any state, and never discards states from open/closed. Consequently, in a finite state space, anchor search is guaranteed to be complete. A full proof by induction under these conditions is found in the literature (Chen and Sturtevant 2019).

Instances of Anchor Search

Enumerating and evaluating all possible instances of the anchor search framework is impractical. We focus on instances that showed reasonable performance in initial, broad experiments. Anchor search instances are categorized according to their anchor and candidate selection policies, and will be evaluated with different types of heuristics.

We refer to a specific anchor search instance as $\text{AS}_{\beta}^{\alpha(k)}$, where α denotes the candidate selection policy, k specifies the number of candidates, and β indicates the anchor selection policy. Considering that the candidate selection policy involves important data structure considerations, we initially divide anchor search instances into three primary classes: *brute-force anchor search* (AS^B), *temporal anchor search* (AS^T), and *randomized anchor search* (AS^R), regardless of the chosen anchor selection policy. A subscript is employed to denote the anchor selection policy. For instance, AS_T^T, AS_A^T, and AS_F^T correspond to AS^T with temporal (T), closest-to-the-opposite-anchor (A), and fixed-to-the-origin (F) anchor selection policies, respectively. Additionally, anchor search instances may utilize different anchor selection policies for the forward and backward frontiers, referred to as hybrid instances. AS_{AF}^T is an AS^T instance where the forward frontier adopts the closest-to-the-opposite-anchor policy, while the backward frontier uses the fixed-to-the-goal policy (i.e., the algorithm expands towards the goal state in the forward direction and towards the anchor of the forward frontier in the backward direction).

Brute-force Anchor Search (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., AS_T^B, AS_R^B, and AS_A^B). This relatively high time complexity renders such AS^B instances infeasible, particularly as the problem complexity and, consequently, the size of the open list increases.

Temporal Anchor Search (AS^T)

Temporal anchor search, is a subclass of anchor search where candidate states are selected according to the order in which they were generated. AS^T uses the k states most recently appended to the open list for constructing the candidate set (for some parameter k). During the expansion phase, the algorithm selects and expands the most promising state among the k most recently generated states. We fix $k = 10$ in our experiments; a study of different values of k is in the appendix. All AS^T instances in our study prioritize states with a higher g-cost for tie-breaking. If a state on the open list is re-generated with a higher g-cost, Temporal Anchor Search retains the node's lower g-cost while treating it as a recently expanded node for temporal prioritization. Doing this improves the overall solution quality.

AS^T has two data structure considerations. First, the temporal order of states added to the open list should be preserved. When removing a state s from the open list (a C++ vector), we swap s with the top element of the open list and then perform a pop operation to remove it.This approach closely approximates the temporal order of the open list and is significantly faster than shifting elements or building a heap. Second, a hash table is used to retrieve the position of states in open; this is used for duplicate detection.

Randomized Anchor Search (AS^R)

In this class of instances, *randomized anchor search*, the candidate set C consists of a subset of k states randomly sampled from the open list 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 AS^R is equivalent to AS^B in terms of expansions. The likelihood of s^* being sampled increases proportionally with k.

Anchor Search and the Baselines

AS^B instances are only viable when the anchors remain fixed or change infrequently during the search. By employing the fixed-to-the-goal anchor selection policy in AS^B, the resultant instance, AS_F^B is 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.

DNR is an instance of 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. A re-sorting (retargeting) of the open list is necessary only upon d-node updates. DNR amortizes the cost of retargeting when the anchor remains constant over many expansions.

TTBS has 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. The set of states evaluated during the mentioned process is similar to a candidate set, and each open list's top state represents the corresponding frontier's anchor.

Front-to-Front Heuristics

The availability of a F2F heuristic is a crucial consideration for employing the anchor search framework effectively. Some domains naturally provide F2F heuristics (e.g., octile distance for grid pathfinding or Manhattan distance for the sliding tile puzzle). In cases where such heuristics are absent, we describe how differential heuristics (DHs) (Sturtevant et al. 2009) can be used to get F2F heuristics estimates from existing F2E heuristics in PDBs.

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. 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. DHs are admissible and consistent on undirected graphs, while ALT (Goldberg and Harrelson 2005) is a similar heuristic for directed graphs.

Pattern databases (PDBs) (Culberson and Schaeffer 1998) map the entire state space V to a more compact state space $\phi(V) = V'$ via a homomorphic abstraction ϕ (Holte et al. 1996). For a given goal $\phi(g) = g' \in V'$, PDBs calculate and store $d(s', g')$ for all $s' \in V'$. A PDB heuristic $h(s,g) = d(\phi(s), g')$ is a lookup in this table. Distances in the PDB are admissible and consistent, and obey the triangle inequality with respect to V'.

We observe that DHs can be applied to abstract distances in PDBs, which is particularly useful when the state space is too large to fit in memory. Instead of building a PDB just to the goal, any pivot $\phi(v) \in V'$ can be used for the PDB. A general F2F PDB DH heuristic is $h(a,b) = |d(\phi(a), \phi(p)) - d(\phi(b), \phi(p))|$.

Lemma 1. *A PDB DH heuristic is both admissible and consistent on an undirected graph.*

Proof. **Admissibility:** Since the distances in the abstract graph (V') obey the triangle inequality, we have:

$$d(\phi(a), \phi(b)) \geq d(\phi(a), \phi(p)) - d(\phi(b), \phi(p))$$
$$d(\phi(b), \phi(a)) \geq d(\phi(b), \phi(p)) - d(\phi(a), \phi(p))$$

Given that the graph is undirected, it follows that:

$$d(\phi(a), \phi(b)) \geq |d(\phi(a), \phi(p)) - d(\phi(b), \phi(p))|$$

As a homomorphic abstraction, the distances between states in V are greater than or equal to the distances in V', leading to $d(a,b) \geq d(\phi(a), \phi(b)) \geq |d(\phi(a), \phi(p)) - d(\phi(b), \phi(p))|$. This proves *admissibility*.
Consistency: To show that the new heuristic is also consistent, we need to demonstrate that for any three states a, b, and c, the inequality $h(a,c) \leq d(a,b) + h(b,c)$ holds. Substituting the DH heuristic into this inequality, we require that both

$$d(\phi(a), \phi(p)) - d(\phi(c), \phi(p)) \leq d(a,b) + d(\phi(b), \phi(p)) - d(\phi(c), \phi(p))$$
$$d(\phi(c), \phi(p)) - d(\phi(a), \phi(p)) \leq d(a,b) + d(\phi(c), \phi(p)) - d(\phi(b), \phi(p))$$

Reordering the terms of the first inequality, we obtain $d(\phi(a), \phi(p)) - d(\phi(b), \phi(p)) \leq d(a,b)$. Since $d(\phi(a), \phi(p)) - d(\phi(b), \phi(p)) \leq d(\phi(a), \phi(b))$ (due to triangle inequality) and $d(\phi(a), \phi(b)) \leq d(a,b)$ (by definition of the abstraction), this inequality holds. Similarly, by reordering the terms of the second inequality, we have $d(\phi(b), \phi(p)) - d(\phi(a), \phi(p)) \leq d(a,b)$. Given that $d(\phi(b), \phi(p)) - d(\phi(a), \phi(p)) \leq d(b,a)$, and since the graph is undirected, this inequality holds as well, establishing that the DH PDB heuristic is consistent. □

Utilizing PDBs as DHs introduces the challenge of pivot placement. 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, and their actual distance is denoted by C^*. 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 C^*. The ideal scenario occurs when $h(a', b')$ equals C^*, 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.

Experimental Results

We broadly compare the performance of anchor search instances with baseline algorithms, including GBFS, BGBFS, DNR with 100 consecutive expansions before retargeting, and two versions of TTBS (TTBS(F) and TTBS(L)) with alternating direction selection policies. The evaluation metrics include state expansions and time, measured on three problem domains: grid pathfinding, the 4×4 sliding tile puzzle (STP), and the 4-peg towers of Hanoi (TOH). While unbounded solution lengths are allowed, we also consider path length as an evaluation metric.

Preliminary experiments showed that AS^T instances perform consistently better than AS^R instances across all the studied domains in this work. Therefore, due to space limitations, detailed results for AS^R are not provided. Among AS^T instances, we focus on three specific instances (AS$_T^{T(10)}$, AS$_A^{T(10)}$, and AS$_{AF}^{T(10)}$), each demonstrating strong performance in at least one domain. We also experimented with a temporal variant of GBFS that is only greedy with respect to recent expansions. This is very similar to a beam search which, like temporal anchor search, does not need a fully sorted open list. But, the results were not strong enough to study this variant more deeply. Note that the strength of TTBS when evaluated previously in planning was diversity, not coverage: instead of solving more problems, it solved problem instances GBFS could not (Kuroiwa and Fukunaga 2021).

TOH and STP experiments were conducted on an Intel Gold 6148 Skylake CPU, 2.4GHz, with 180GB of available memory. For grid pathfinding, an Intel E5-2683 v4 Broadwell CPU, running at 2.1GHz, and featuring 250GB of available memory, was utilized.

Grid Pathfinding

We experimented on four sets of 8-connected maps from the MovingAI repository (Sturtevant 2012): 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 AS^T

	Anchor Search			Baselines			
Maps	$\mathbf{AS}_T^{T(10)}$	$\mathbf{AS}_A^{T(10)}$	$\mathbf{AS}_{AF}^{T(10)}$	**GBFS**	**BGBFS**	**DNR**	**TTBS(L)**
				Expansions			
DAO	**3,809**	6,586	5,893	4,310	5,283	4,266	4,177
SC1	9,174	16,159	14,525	11,293	12,205	10,983	**8,952**
WC3	1,330	2,660	2,045	1,887	1,348	1,862	**1,252**
Mazes	**41,509**	52,827	50,325	45,694	49,327	47,156	43,496
				Time (ms)			
DAO	**1.91**	3.53	2.99	9.10	19.93	22.35	20.47
SC1	**5.53**	10.13	8.63	25.37	50.09	67.54	44.53
WC3	**0.75**	1.49	1.13	7.40	16.96	8.31	5.21
Mazes	**21.31**	30.33	26.48	40.90	61.38	343.42	210.01
				Solution Length			
DAO	499	481	489	**473**	**477**	486	581
SC1	746	706	731	690	**684**	729	933
WC3	412	391	416	387	**381**	404	470
Mazes	2,289	2,248	2,257	2,270	2,272	**2,217**	2,487

Table 1: Results for grid pathfinding.

instances, which is much faster than a general hash table; diagonal ($\sqrt{2}$) edge costs precluded additional optimizations.

The results, presented in Table 1, underscore the robust performance of AS_T^T. It notably outperformed all five baseline algorithms in expansions for the DAO and mazes benchmarks, with slightly worse performance than TTBS(L) in SC1 and WC3. All AS^T algorithms exhibited a statistically significant improvement in execution time (95% confidence interval), with an order of magnitude reduction compared to the baseline methods due to the constant time complexity of expansion. Importantly, the solution lengths achieved by AS^T instances also remained on par with those of the baselines. TTBS(F) is omitted because the results are nearly identical to TTBS(L).

4-peg Towers of Hanoi (TOH)

We next evaluate anchor search in TOH. In TOH, algorithms are typically run to the standard goal state (Chen et al. 2017); we take up the more challenging problem of searching between random states. This is challenging because solution symmetry and large pre-computed heuristics, which were necessary to scale optimal solutions to 30 disks (Korf and Felner 2007), cannot be exploited. TOH lacks a natural F2F heuristic, and thus makes a good testbed for assessing the effectiveness of PDB DHs.

The settings used for these experiments were found by running preliminary experiments on TOH(10) to evaluate the best anchor search instances. This included an evaluation of the size of the candidate set, showing that 10 candidates performed best overall. We also evaluated GBFS with a temporal expansion policy, something we discuss at the end of the section when summarizing the impact of the bidirectional nature of anchor search.

In all experiments, we utilized DH 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. This PDB takes 12.9ms to build on average, which is not included in the reported running time. 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-∞).

The 12-disk PDBs take 6.66s to build, on average, which, again, is not included in running times. 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 weighting PDB(0-11), we aimed to preserve the overall heuristic gradient (at the cost of losing admissibility). The second lookup has the largest impact 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 experimental results demonstrated the opposite in unbounded suboptimal settings, consistent with previous findings (Wilt and Ruml 2016). We will refer to TOH with n disks as TOH(n).

22-24 disk experiments We then narrowed our attention to the top-performing instances, namely AS_{AF}^T, BGBFS, and GBFS. They were employed to solve TOH problems

Metrics	22 disks			24 disks		
	$AS_{AF}^{T(10)}$	GBFS	BGBFS	$AS_{AF}^{T(10)}$	GBFS	BGBFS
DH PDB(0-11)						
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
DH PDB(0-11)×100 + DH 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
DH PDB(0-11)×100 + DH 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	**2,397**

Table 2: TOH results for the differential heuristics

Metrics	DH PDB(12) + 180GB		DH PDB(15) + 700GB	
	26 disks	28 disks	30 disks	32 disks
Exp.	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 3: $AS_{AF}^{T(10)}$ solving TOH problems with 26—32 disks using 180GB and 700 GB of RAM.

with 22 and 24 disks. In this experiment (Table 2), when utilizing single PDB lookups as DHs (the first setting), AS_{AF}^{T} 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 AS_{AF}^{T} on average for the problem sizes of 22 and 24, respectively. Meanwhile, BGBFS expanded 2.5 and 3.5 times as many states as AS_{AF}^{T} for the same problem sizes. Regarding average runtime, AS_{AF}^{T} was between 1.9 and 2.0 times faster than GBFS and BGBFS in TOH(22), and 3.0 and 3.6 times faster in TOH(24). AS_{AF}^{T} also strongly outperformed GBFS and BGBFS in terms of solution length. Comparing the average expansions and solution length of AS_{AF}^{T} in this experiment, it can be seen that the paths found contain more than 70% of the states expanded. This demonstrates that 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 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 AS_{AF}^{T} still demonstrated fewer average expansions, the trend shifted in favor of GBFS and BGBFS in 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).

Notably 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.

26-32 disk experiments The strong performance of AS_{AF}^{T} in the second experiment motivated us to conduct another experiment, aimed at solving larger TOH problems with 26, 28, 30, and 32 disks.

In this experiment (Table 3), 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. This memory is used for storing the open and closed lists during search. This PDB took 460.6s on average to build, which is not included in the runtimes. 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)

	Anchor Search			Baselines				
Metrics	$\mathbf{AS}_{T}^{T(10)}$	$\mathbf{AS}_{A}^{T(10)}$	$\mathbf{AS}_{AF}^{T(10)}$	**GBFS**	**BGBFS**	**DNR**	**TTBS(L)**	**TTBS(F)**
Expansions	92,907	**1,186**	1,449	1,945	2,317	**1,003**	56,475	2,156
Time (ms)	181.04	**2.13**	2.54	3.43	5.50	8.25	288.63	9.45
Length	12,907	380	460	463	311	361	49,084	**269**

Table 4: STP results on 100 problems instances.

and TOH(32). However, $\text{AS}_{AF}^{T(10)}$ successfully solved all the given problem instances with 26, 28, 30, and 32 disks, with results in Table 3.

TOH Summary In TOH, it can be observed that the expansion rate of anchor search instances is slower than 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 complex heuristics. The efficiency of heuristic computations has a greater impact on anchor search instances than on GBFS, as the former involves performing more heuristic evaluations.

Our results suggest that the bidirectional nature of AS_{AF}^{T} is important when solving large TOH instances. 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. Our tests with the temporal variant of GBFS also did not outperform anchor search. This reinforces that the bidirectional and temporal aspects of anchor search both contribute to overall performance.

Sliding Tile Puzzle (STP)

Finally, we perform experiments on the Sliding-Tile puzzle. We again searched between arbitrary start and goal states, instead of using the canonical goal state.

Table 4 shows the average expansions, time, and solution length of 100 problems with i^{th} and $i+10^{\text{th}}$ Korf's instances (Korf 1985) 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 AS_{A}^{T} (1186 ± 130) and DNR (1003 ± 123) highlighted the insignificant difference in their performance in terms of expansions. Moreover, AS_{A}^{T} had the best runtime, while DNR achieved the highest average solution quality. Notably, 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.

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 (Kuroiwa and Fukunaga 2021). In the anchor search framework, AS_{AF}^{T} is not only robust, but also consistently outperforms GBFS. With regard to anchor search instances, 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 (Imai and Kishimoto 2011), and suggests why AS_{AF}^{T} is the most robust instance explored.

Conclusions and Future Work

This work describes the anchor search framework for F2F suboptimal Bi-HS. We evaluated this framework through three specific anchor search instances in various domains and settings, achieving promising performance against the baselines. We also proposed the idea of using PDBs as DHs, which makes F2F search algorithms applicable in domains without natural F2F heuristics. This work provides a foundation for future approaches in suboptimal bidirectional search, such as modifying these algorithms to find solutions with bounded suboptimality.

Acknowledgments

This work was supported by the National Science and Engineering Research Council of Canada Discovery Grant Program and the Canada CIFAR AI Chairs Program. This research was enabled in part by support provided by Prairies DRI and the Digital Research Alliance of Canada (alliancecan.ca). This work was also supported by the Israel Science Foundation (ISF) grant #909/23 awarded to Shahaf Shperberg and Ariel Felner, by Israel's Ministry of Innovation, Science and Technology (MOST) grant #1001706842, in collaboration with Israel National Road Safety Authority and Netivei Israel, awarded to Shahaf Shperberg, and by BSF grant #2024614 awarded to Shahaf Shperberg.

References

Alcázar, V. 2021. 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 *ICAPS*, 7–15.

Atzmon, D.; Shperberg, S. S.; Sabah, N.; Felner, A.; and Sturtevant, N. R. 2023. W-restrained Bidirectional Bounded-Suboptimal Heuristic Search. In *ICAPS*, 26–30. AAAI Press.

Bisiani, R. 1992. Beam search. *Encyclopedia of artificial intelligence*.

Björnsson, Y.; Bulitko, V.; and Sturtevant, N. R. 2009. TBA*: Time-Bounded A*. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 431–436.

Chen, J.; Holte, R. C.; Zilles, S.; and Sturtevant, N. R. 2017. Front-to-End Bidirectional Heuristic Search with Near-Optimal Node Expansions. *International Joint Conference on Artificial Intelligence (IJCAI)*.

Chen, J.; and Sturtevant, N. R. 2019. Conditions for Avoiding Node Re-expansions in Bounded Suboptimal Search. *International Joint Conference on Artificial Intelligence (IJCAI)*.

Culberson, J. C.; and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence*, 14(3): 318–334.

de Champeaux, D.; and Sint, L. 1977. An Improved Bidirectional Heuristic Search Algorithm. *J. ACM*, 24(2): 177–191.

Dechter, R.; and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A. *Journal of the ACM (JACM)*, 32(3): 505–536.

Eckerle, J.; Chen, J.; Sturtevant, N.; Zilles, S.; and Holte, R. 2017. Sufficient Conditions for Node Expansion in Bidirectional Heuristic Search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 79–87.

Felner, A.; Moldenhauer, C.; Sturtevant, N. R.; and Schaeffer, J. 2010. Single-Frontier Bidirectional Search. *AAAI Conference on Artificial Intelligence*, 59–64.

Goldberg, A. V.; and Harrelson, C. 2005. Computing the shortest path: A search meets graph theory. In *SODA*, volume 5, 156–165.

Hansen, E. A.; and Zhou, R. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research*, 28: 267–297.

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

Heusner, M.; Keller, T.; and Helmert, M. 2017. Understanding the search behaviour of greedy best-first search. In *Proceedings of the International Symposium on Combinatorial Search*, 1, 47–55.

Holte, R. C.; Felner, A.; Sharon, G.; and Sturtevant, N. R. 2016. Bidirectional Search That Is Guaranteed to Meet in the Middle. In *AAAI*, 3411–3417. AAAI Press.

Holte, R. C.; Perez, M. B.; Zimmer, R. M.; and MacDonald, A. J. 1996. Hierarchical A*: Searching abstraction hierarchies efficiently. In *AAAI/IAAI, Vol. 1*, 530–535.

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

Islam, F.; Narayanan, V.; and Likhachev, M. 2016. A*-Connect: Bounded suboptimal bidirectional heuristic search. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2752–2758. IEEE.

Kaindl, H.; and Kainz, G. 1997. Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research*, 7: 283–317.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1): 97–109.

Korf, R. E.; and Felner, A. 2007. Recent Progress in Heuristic Search: A Case Study of the Four-Peg Towers of Hanoi Problem. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 20, 2324–2329. Citeseer.

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

Likhachev, M.; and Stentz, A. 2008. R* search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 344–350.

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

Politowski, G.; and Pohl, I. 1984. D-Node Retargeting in Bidirectional Heuristic Search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 274–277.

Shaham, E.; Felner, A.; Sturtevant, N. R.; and Rosenschein, J. S. 2018. Minimizing Node Expansions in Bidirectional Search with Consistent Heuristics. In *Proceedings of the International Symposium on Combinatorial Search*, 81–89.

Shperberg, S.; Felner, A.; Sturtevant, N. R.; Hayoun, A.; and Shimony, E. S. 2019. Enriching Non-parametric Bidirectional Search Algorithms. In *AAAI*, 2379–2386.

Siag, L.; Shperberg, S. S.; Felner, A.; and Sturtevant, N. R. 2023. Comparing Front-to-Front and Front-to-End Heuristics in Bidirectional Search. In *SOCS*, 158–162. AAAI Press.

Sint, L.; and De Champeaux, D. 1977. An improved bidirectional heuristic search algorithm. *Journal of the ACM (JACM)*, 24(2): 177–191.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2): 144–148.

Sturtevant, N. R.; Felner, A.; Barer, M.; Schaeffer, J.; and Burch, N. 2009. Memory-based heuristics for explicit state spaces. *International Joint Conference on Artificial Intelligence (IJCAI)*, 609–614.

Sturtevant, N. R.; Shperberg, S.; Felner, A.; and Chen, J. 2020. Predicting the Effectiveness of Bidirectional Heuristic Search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 281–290.

Valenzano, R.; Arfaee, S. J.; Stern, R.; Thayer, J.; and Sturtevant, N. 2013. Using Alternative Suboptimality Bounds in Heuristic Search. In *ICAPS*, 233–241.

Wilt, C.; and Ruml, W. 2016. Effective heuristics for suboptimal best-first search. *Journal of Artificial Intelligence Research*, 57: 273–306.

Wilt, C.; Thayer, J.; and Ruml, W. 2010. A comparison of greedy search algorithms. In *Proceedings of the International Symposium on Combinatorial Search*, 1, 129–136.