Bidirectional Search for Longest Paths: Case for Front-to-Front Heuristics

Tzur Shubi, Ariel Felner, Solomon Eyal Shimony, Shahaf S. Shperberg

Stein Faculty of Computer and Information Science, Ben-Gurion University of the Negev, Israel

Abstract

Bidirectional heuristic search can potentially reduce search effort for problems amenable to backward search. Therein, it is well-known that front-to-front heuristics can reduce the number of node expansions, but their overhead is so high that overall runtime almost always increases. We propose BiXDF-BnB, a bidirectional depth-first branch-and-bound algorithm that adapts the Single-Frontier Bidirectional Search (SFBDS) framework—originally developed for shortest-path (MIN) problems—to the Generalized Longest Simple Path (GLSP) setting. Because SFBDS inherently operates on paired states, front-to-front (F2F) heuristic evaluation arises naturally and avoids the overhead typically associated with bidirectional frontier management. We show that this adaptation can be successfully applied to maximization (MAX) problems while efficiently handling overlapping constraints. BiXDFBnB is applied to several types of longest-path problems: Longest Simple Path (LSP), Snakes, and Coil-in-the-Box (CIB). Empirical evaluation shows that the new algorithm frequently reduces the number of node expansions and, in some cases, also improves overall runtime.

1 Introduction

Bidirectional heuristic search can potentially reduce search effort for combinatorial search problems amenable to backward search. Therein, it is well-known that front-to-front heuristics can reduce the number of expansions in search, but their overhead is so high that overall runtime almost always increases. This paper examines whether front-to-front heuristics can be beneficially applied in longest-path search.

The aim in the *shortest path problem* is to minimize a path cost between two given vertices. Such problems are called *minimization* (MIN) problems. A prominent problem in graph theory is to find the *longest simple path* (LSP) between two given vertices in an undirected graph. A *simple path* is one where no vertex is visited more than once. In LSP, the aim is to find a maximum-cost path. Such problems are called *maximization* (MAX) problems. LSP is known to be NP-hard, and even hard to approximate within a constant factor (Karger, Motwani, and Ramkumar 1997). The motivation for solving LSP comes from a variety of domains such as information retrieval on peer-to-peer networks (Wong, Lau, and King 2005), estimating the worst packet delay of Switched Ethernet network (Schmidt and Schmidt 2010), multi-robot patrolling (Portugal and Rocha 2010), and VLSI design where the longest path should be found between two components on a printed circuit board (Chen 2016). Another important variant of LSP is *Snake-in-the-box* (SIB) (Kautz 1958). In a *snake*, a path may not use neighbors of vertices that are already in the path. A *box* is an n-dimensional cube. SIB can help identify efficient error-correcting codes.

Several prior works treated LSP as a heuristic search problem: Stern et al. (2014) modified heuristic search algorithms designed for MIN problems to solve MAX problems. Follow-up work (Cohen, Stern, and Felner 2020) proposed refinements, including grid-based heuristics for LSP, and introduced several admissible heuristics for the SIB problem (Palombo et al. 2015).

LSP and SIB were recast within a *generalized longest simple path* (GLSP) framework that includes these problems as special cases (Dahan et al. 2022). Most prior work on GLSP consists of unidirectional heuristic search algorithms, such as A* and branch-and-bound, that were modified to work with MAX problems. Recent work (Shubi et al. 2025) proposed XMM, a bidirectional search algorithm for GLSP that uses front-to-end heuristics in an attempt to approximate a meet-in-the-middle (MM) approach (Holte et al. 2017). Unlike in MIN problems, where lower-bound heuristics enable algorithms to prevent the expansion of nodes beyond the optimal solution's midpoint (*restrained search*), bidirectional search for GLSP (MAX) relies on overestimating upper bounds. This makes it hard to identify exactly when the search frontiers have crossed the midpoint.

In bidirectional GLSP search, when forward and backward frontiers meet, one must verify that the concatenated paths are valid (exclusion constraints like vertex overlaps not violated). Maintaining separate frontiers and validating all crossing path pairs incurs large overhead. To address this, we introduce BiXDFBnB, a depth-first bidirectional branch-and-bound algorithm that restructures the Single-Frontier Bidirectional Search (SFBDS) framework (Felner et al. 2010)—originally designed for MIN problems—to conquer the challenges of the GLSP (MAX) setting.

Instead of growing two independent frontiers, BiXDF-BnB maintains just one pair of (forward, backward) states per search node. Given this synchronized paired-state struc-

ture, utilizing a pairwise front-to-front (F2F) heuristic is the natural design choice, entirely eliminating the frontier-matching bottleneck. However, porting this architecture from MIN to MAX is non-trivial due to the reliance on over-estimating upper bounds and the inability to prune beyond a guaranteed midpoint. A core algorithmic novelty of this paper is in effectively utilizing F2F bounds to prune the MAX search space. In addition, we integrate simultaneous Cartesian node expansion to guarantee valid path-meetings without parity issues. We then provide theoretical guarantees on the structural properties of the resulting algorithm followed by empirical evaluation validating its advantages.

An additional contribution is that for unit edge costs, BiXDFBnB "meets in the middle", a property not achievable in XMM. An empirical evaluation strongly supports the case for F2F heuristics in BiXDFBnB. In many cases, BiXDFBnB also outperforms XMM and unidirectional search.

2 Background

2.1 Generalized Longest Simple Path Problems

The Generalized Longest Simple Path (GLSP) framework (Dahan et al. 2022) includes both LSP and Snakes. Let $G = (V, E, w)$ be a connected undirected weighted graph. A path from v_0 to v_m is an alternating sequence $P = (v_0, (v_0, v_1), v_1, (v_1, v_2), \ldots, v_m)$ of vertices and edges in G such that consecutive elements in the sequence are adjacent, i.e., each edge is incident on the preceding and following vertices in P.

A (global) *exclusion constraint* L is a function mapping each $x \in (V \cup E)$ to a subset of $V \cup E$. The constraint means that **after** exiting x, a path may not visit any member of $L(x)$. For example, the constraint: $\forall x \in V,\ L(x) = \{x\}$. i.e. "no vertex may be visited more than once" defines the longest simple path (LSP) problem (Elements x for which $L(x)$ is not specified - edges here - are assumed to have no constraint). To define *snake problems* (not necessarily "in a box"), we have: $\forall x \in V,\ L(x) = \{x\} \cup N(x)$, where $N(x)$ are the immediate neighbors of x.

Definition 1 (Generalized LSP Problem)**.** *Given a graph $G = (V, E, w)$, and a constraint L, find a path of maximum weight w in G (optionally starting at start vertex s, optionally ending at target vertex t) that does not violate L.*

In this paper, we only examine the case where G is unweighted ($w(e) = 1 \quad \forall e \in E$), and where the path must start at s and end at t.

2.2 Best-First Exhaustive Search in GLSP

A *state* in the state-space is a path π beginning at start vertex s. We assume, unless stated otherwise, that forward search begins with start state $\pi = (s)$, and progresses by adding one edge and vertex at a time to a path π that has s' as a last vertex. A *goal state* is a path ending at target vertex t. Following Stern et al. (2014), for a search node N, we denote its state, i.e., the (partial) path it represents, by $N.\pi$, its most recently added vertex s' by $N.head$, and the rest of the vertices in $N.\pi$ by $N.tail$. For standard unidirectional search, $g(N)$ is naturally the length of $N.\pi$. The successors of $N.\pi$ are denoted by $\Gamma(N.\pi)$. The *remaining graph* $G_r(G, N.\pi)$ is defined to be G after removing (1) all $N.\pi$ vertices other than $N.head$ from G, and (2) all other elements no longer allowed under constraint L. Clearly, a node N is not a dead end only if $N.head$ and t are in the same connected component of $G_r(G, N.\pi)$.

In shortest-path problems (and in MIN problems in general), search algorithms such as A* prioritize nodes with lower $f(N) = g(N) + h(N)$ values, where g is the path cost from the start state to N, and h is a cost estimation of the **shortest** path from N to the goal. Stern et al. (2014) present modifications that need to be done in order to modify A* (commonly used for MIN problems) to MAX problems. It is common practice to use all these modifications when implementing A* for MAX problems. A prominent modification for MAX is that A* now prioritizes nodes with the largest $f(N)$ values, as done specifically for LSP, where h in GLSP estimates the length of the **longest** path to the goal.

In GLSP, duplicate states do not occur because the state is a path, rather than just a head vertex. Still, one can remove symmetric states: in LSP states with the same head vertex, and the same set of visited vertices (but in a different order) (Cohen, Stern, and Felner 2020).

A function h is said to be *admissible* for GLSP (i.e., MAX problems) iff for every node N in the search space, $h(N)$ is *greater than or equal to* the weight of the longest constrained path (longest simple path in LSP) from $N.head$ to t in $G_r(G, N.\pi)$. GLSP heuristics use graph connectivity (Dahan et al. 2022; Palombo et al. 2015). Trivially, an admissible heuristic (upper bound) is the entire remaining graph G_r size. Obviously, it is also admissible to delete vertices no longer reachable in $G_r(G, N.\pi)$ from the path head $N.head$ or from the target t. Finally, one can delete vertices $v \in G_r(G, N.\pi)$ for which there is no simple path from $N.head$ to t containing v. This heuristic is denoted by h_{BCC} (Dahan et al. 2022) since it can be efficiently computed using biconnected components as follows: Let $G_r^+(G, N.\pi) = G_r(G, N.\pi) \cup (N.head, t)$, and let B be the biconnected block of $G_r^+(G, N.\pi)$ containing $N.head$ and t. Then v is a vertex on some simple path from $N.head$ to t in $G_r(G, N.\pi)$ iff v is in B. The number of edges in a simple path from $N.head$ to t is thus at most $h_{BCC} = |V(B)| - 1$, where $V(B)$ are the vertices of B. A stronger heuristic (h_{MIS}) based on maximum independent sets and triconnectivity exists (Dahan et al. 2022, 2024). As h_{MIS} was ineffective for highly connected graphs, as used in our experiments, this heuristic is not further discussed here.

Snake problems have stronger heuristics (Palombo et al. 2015). For example, the *snake head* heuristic (h_{SH}) recognizes that only one neighbor of the head can be used in a snake path. Another heuristic is the *star pattern* heuristic (Palombo et al. 2015), which subtracts 1 from the bound for each vertex and all its (≥ 3) neighbors not yet visited. This heuristic was improved by Dahan et al. (2022) to a vertex and only 3 of its neighbors, called the "*Y* heuristic" (h_Y). We use the best-performing heuristic in our evaluation.

Unidirectional DFBnB for GLSP Unidirectional Depth-First Branch and Bound (DFBnB) for longest path (denoted

here by XDFBnB) (Stern et al. 2014) iteratively extends a single path starting from the initial vertex s. It maintains an incumbent longest s-t path, S, found so far. At each step, an admissible heuristic upper bound on the remaining path length is added to the current path's length. If this sum is not strictly greater than $|S|$, the current branch cannot improve the incumbent and is safely pruned. Otherwise, the valid successors of the current path's head are generated, sorted in descending order of their path cost estimates $f = g + h$, and recursively expanded. This node ordering encourages finding long paths early, thus rapidly tightening the lower bound $|S|$ and pruning remaining branches more effectively.

2.3 Bidirectional Search

In BiHS for MIN problems, the objective is to find a least-cost path between $start$ and $goal$ in a given graph G. The shortest distance between nodes x and y is denoted by $c(x, y)$, where $c(start, goal) = C^*$. BiHS performs a forward search (F) from $start$ and a backward search (B) from $goal$, continuing until the two search frontiers meet. BiHS algorithms typically maintain two open lists, OPEN_F and OPEN_B, for the forward and backward searches, respectively. Each node is associated with g-, h-, and f-values. For a given direction $D \in \{F, B\}$, these values are denoted as g_D, h_D, and f_D (i.e., g_F, h_F, f_F for forward search and g_B, h_B, f_B for backward search).

Most BiHS algorithms utilize two *front-to-end* (F2E) heuristic functions (Kaindl and Kainz 1997), $h_F(s)$ and $h_B(s)$, which estimate $c(s, goal)$ and $c(start, s)$ for all $s \in G$, respectively. A forward heuristic h_F is said to be *admissible* if $h_F(s) \leq c(s, goal)$ for all s in G, and *consistent* if $h_F(s) \leq c(s, s') + h_F(s')$ for all s and s' in G. Analogous definitions apply for backward admissibility and consistency. More informative are *front-to-front* (F2F) heuristics, which instead estimate the distance $c(s, s')$ between a given node s on the forward frontier to each node s' on the backward frontier by $h(s, s')$. A typical priority function here is then given by:

$$f_F(s) = g_F(s) + \min_{s' \in O_B} (h(s, s') + g_B(s'))$$

Since the heuristic estimations in this $f_F(s)$ is for smaller parts (between two frontier nodes) it tends to be more accurate than front-to-end heuristics (which estimate all the way towards the goal). Unfortunately, despite drastically decreasing the number of expansions, the overhead due to having to compute the minimum over all the nodes in the opposite frontier typically increases the overall runtime. Thus, front-to-front heuristics are rarely used in practice.

Single frontier bidirectional search (SFBDS) (Felner et al. 2010) is a general front-to-front BiHS framework which only uses a single frontier. SFBDS basically spans a search tree that can then be searched with any known search algorithm. A node in a search tree spanned by SFBDS consists of a pair of states (x, y): a forward state x and a backward state y. Such a node corresponds to the task of finding a path between x and y. This task is recursively decomposed by expanding either x or y and generating new tasks between either (1) $\Gamma(x)$ and y, or (2) $\Gamma(y)$ and x. At every node, an *expansion policy* (also called *jumping policy*) decides which of the two states to expand next, i.e., the search can proceed forward or backward. Given a fixed expansion policy, a tree is induced where the cost of a node in that tree is $f(x, y) = g_F(x) + g_B(y) + h(x, y)$, where $h(x, y)$ is a F2F heuristic between x and y. That tree can be searched using any admissible search algorithm, such as A* or IDA*.

2.4 Bidirectional Search in GLSP

BiHS requires the ability to perform backward search from the goal state(s), as well as forward search from the initial state. However, in GLSP, a state consists of an entire path. To achieve backward search for GLSP, our backward search mirrors the forward search by growing an initially empty path starting from the target *vertex* t rather than from a goal *state*. Such backward search creates difficulties when aiming at bidirectional search - challenges that have been addressed by Shubi et al. (2025), as follows.

Solution Detection. In bidirectional search for GLSP, algorithms must cross-reference frontiers to find path intersections (e.g., by indexing nodes by their $head$ vertex). Once a match is found, the algorithm must perform *path verification* to ensure the concatenated paths do not overlap or violate any GLSP constraints before a valid solution is declared.

Search Bounds. When to halt the search is another non-trivial issue. In bidirectional best-first algorithms for MAX problems (like XMM), the search maintains a global upper bound on the longest path based on the heuristic evaluations of the forward and backward frontiers. The search proves optimality and halts when it finds a path whose cost equals this upper bound. In contrast, depth-first branch-and-bound approaches do not maintain expansive frontiers; instead, they maintain a global incumbent solution and prune any individual branch whose heuristic upper bound cannot strictly improve upon the incumbent.

2.5 XMM and Bidirectional Baselines

A prominent strategy in standard BiHS is attempting to "meet in the middle" by delaying the expansion of nodes that have progressed past the solution midpoint (Holte et al. 2017). For MAX problems like GLSP, the state-of-the-art baseline is *MM for Maximum*, **XMM** (Shubi et al. 2025), which achieves this by ordering its open lists using the following priority function:

$$pr_D(n) \triangleq \min(2h_D(n), g_D(n) + h_D(n)) \qquad (1)$$

This function behaves like standard $f = g + h$ initially, but safely delays the expansion of nodes where $g_D(n) > h_D(n)$ (nodes estimated to be past the middle). While the overestimating nature of MAX heuristics prevents XMM from strictly guaranteeing that no nodes beyond the midpoint are expanded, it serves as the primary best-first bidirectional baseline for our empirical evaluation.

3 Bidirectional DFBnB for GLSP

To overcome the limitations of standard bidirectional architectures in maximization settings, we introduce BiXDF-BnB. This algorithm represents a non-trivial realization of

Algorithm 1: BiXDFBnB: Bidirectional DFBnB for Longest Simple Path

Input : Undirected graph $G = (V, E)$, start $s \in V$, target $t \in V$, admissible front-to-front heuristic h_{F2F}
Output: A longest simple s-t path S

```
1  global S ← ∅
2  BiXDFBnB(MkNd(s), MkNd(t))
3  return S
4  Function BiXDFBnB(s_F, s_B):
5      if |s_F.π| + h_F2F(s_F, s_B) + |s_B.π| ≤ |S| then
6          return                                  // Pruning
7      if s_F.head = s_B.head then
8          π ← s_F.π · reverse(s_B.π)              // Exact Meet
9          if |π| > |S| then
10             S ← π
11         return
12     else if s_F.head ∈ s_B.tail or s_B.head ∈ s_F.tail then
13         return                                  // Overlap Rejection
14     else if (s_F.head, s_B.head) ∈ E then
15         π ← s_F.π · reverse(s_B.π)              // Adjacent Meet
16         if |π| > |S| then
17             S ← π
                                                   // Simultaneous Expansion
18     foreach (succ_F, succ_B) ∈ Sorted(Γ(s_F) × Γ(s_B)) do
19         BiXDFBnB(succ_F, succ_B)                // Empty product prunes dead-ends
```

the Single-Frontier Bidirectional Search (SFBDS) framework (Felner et al. 2010) engineered specifically for the complex constraints of GLSP. While SFBDS was originally conceived to minimize frontier matching overhead in shortest-path (MIN) problems, transitioning it to a maximization (MAX) depth-first branch-and-bound framework requires a fundamental rethink of node representation, expansion coordination, and pruning mechanics. Rather than simply tracking independent frontiers, BiXDFBnB structuralizes the search space into tightly synchronized paired states, turning what is traditionally a costly frontier-coordination liability into an algorithmic advantage for tracking path constraints.

3.1 The BiXDFBnB Algorithm

Algorithm 1 outlines the pseudocode for BiXDFBnB. The algorithm maintains the incumbent longest path found so far, denoted by S, initialized to an empty path (line 1). Instead of maintaining expanding open lists, at each recursive step, the search evaluates a single pair of states: a forward state s_F representing a path from s, and a backward state s_B representing a path from t. The recursive call has three primary phases: pruning, path overlap rejection, and expansion.

Bound Pruning (Lines 5-6): The algorithm relies on an admissible front-to-front heuristic $h_{F2F}(s_F, s_B)$, which provides an upper bound on the length of the longest simple path connecting $s_F.head$ to $s_B.head$ using only remaining valid vertices. In Line 5, the algorithm evaluates the pruning condition. If the length of the forward path plus the backward path, plus the heuristic estimate is less than or equal to the length of the incumbent solution $|S|$, the current branch cannot possibly yield a strictly longer path. The algorithm thus safely prunes the node and backtracks.

Pruning overlapping paths (Lines 12-13): To guarantee path compliance with the global exclusion constraint L, the algorithm verifies that the head of one state has not collided with the previously visited body ($tail$) or other constrained vertices of the opposing state. If such an intersection occurs, the paths cross illegally, and the branch is pruned.

Node Expansion Strategy (Lines 18-19): The canonical dynamic of SFBDS alternates between expanding in one direction by a single vertex while freezing the other. However, BiXDFBnB opts for a *Simultaneous Expansion* paradigm. Rather than choosing one side to advance, it expands both paths simultaneously by generating the Cartesian product of the valid successor sets: $\Gamma(s_F) \times \Gamma(s_B)$. For each pair $(succ_F, succ_B)$ generated by this Cartesian product, the algorithm computes the front-to-front heuristic $h_{F2F}(succ_F, succ_B)$. The pairs are sorted in descending order according to these heuristic estimates and explored via depth-first recursive calls.

Due to this expansion scheme, there are two possible path meeting scenarios, as we next explain.

Exact Meet (Lines 7-11): If both paths terminate on the exact same vertex, a valid s-t path is formed by concatenating $s_F.\pi$ with the reversed $s_B.\pi$. If this new path improves upon S, the incumbent is updated. Because the paths have collided, no further simple extensions are possible, and the algorithm backtracks.

Adjacent Meet (Lines 14-17): Advancing both paths simultaneously induces the *Adjacent Meet* condition: the forward and backward path heads can become adjacent in G before landing on the exact same vertex. In this scenario, a valid s-t path is successfully bridged via the connecting edge, and the incumbent S is updated if this bridged path is longer. Simultaneous expansion effectively collapses two independent alternating steps into one, tightly coupling the front-to-front heuristic evaluation of the joint moves while naturally catching valid paths before the search frontiers illegally cross.After an adjacent meet the BiXDFBnB does not backtrack. Indeed, it prevents both searches from continuing via the crossing edge. However, because new longer paths can still be formed, the search continues via other edges.

3.2 Generalizing to GLSP

While Algorithm 1 focuses on the Longest Simple Path (LSP) problem, BiXDFBnB naturally generalizes to all Generalized Longest Simple Path (GLSP) framework problems. Simply replacing the overlap rejection conditions with the appropriate GLSP constraint check suffices. However, some

optimizations become possible as well. To demonstrate this adaptability, we use the Snake problem as a representative example, requiring only two minor adjustments:

General Overlap Rejection For LSP, paths must avoid the opposing visited body ($tail$). Because Snake requires an induced (chordless) path, the overlap rejection condition (Line 12) must be broadened. The head of one state must not land on any *illegal* vertices of the opposing state, which includes both the opposing $tail$ and all of its neighbors.

Immediate Backtracking on Adjacent Meets In LSP, the search continues after an adjacent meet to find potentially longer detours. For Snake, however, the algorithm must backtrack immediately by inserting a **return** statement after line 17, since any further extensions would instantly yield an illegal chord. Therefore, after updating the incumbent, no strictly longer valid snake can exist from these prefixes, and the branch should be pruned.

3.3 Theoretical Properties

A crucial property of BiXDFBnB is that, because it uses a depth-first search strategy, its memory usage is linear in the search depth d. Specifically, the memory complexity is $O(d \cdot b^2)$, where b is the maximum branching factor. This provides a significant advantage over algorithms that maintain explicit open lists, whose memory requirements can scale exponentially.

Next, we establish the soundness, completeness, and optimality of BiXDFBnB, as well as its meet-in-the-middle property. A bidirectional algorithm is particularly vulnerable to generating cyclic paths or bypassing the optimal solution due to search frontiers crossing illegally. We show that the synchronized Cartesian expansion inherently resolves these parity issues.

Lemma 1 (Soundness)**.** *BiXDFBnB only generates valid, simple s-t paths.*

Proof. Candidate paths are only formed under two mutually exclusive conditions: Exact Meet and Adjacent Meet. Suppose a forward path π_F and a backward path π_B share a vertex v. Because the search expands both frontiers simultaneously via the Cartesian product, the paths grow at an identical rate: $|\pi_F| = |\pi_B| = k$ at any search depth k. Three distinct cases arise:

(1) If both paths arrive at v at the exact same depth k, the algorithm triggers an *Exact Meet* ($s_F.head = s_B.head$), safely merges the paths, and backtracks, preventing further expansion.

(2) If the frontiers try to cross each other by traversing the same edge in opposite directions simultaneously, they must have been positioned on adjacent vertices at the previous depth $k-1$. This scenario is perfectly intercepted by the *Adjacent Meet* condition, avoiding illegal intersections.

(3) Alternatively, if π_F reaches v at depth k_1 and π_B reaches v at depth k_2 where $k_1 < k_2$, then v becomes part of $s_F.tail$. When the backward search lands on v at depth k_2, the *Overlap Rejection* condition ($s_B.head \in s_F.tail$) is triggered, immediately pruning the branch.

Consequently, it is topologically impossible to concatenate two overlapping paths, ensuring all generated paths are strictly simple. □

Lemma 2 (Intersection Guarantee)**.** *For any valid simple s-t path P, the simultaneous expansion strategy guarantees the search frontiers will trigger an Exact Meet or Adjacent Meet, provided the branches are not pruned.*

Proof. Let P be a simple s-t path of length L (consisting of L edges). Because the Cartesian product $\Gamma(s_F) \times \Gamma(s_B)$ is explored at each step, the algorithm synchronously evaluates prefixes and suffixes of P, expanding by exactly one edge per prefix and one edge per suffix at a time.

Even Length ($L = 2k$): P contains an exact middle vertex. At search depth k, both s_F and s_B will have traversed exactly k edges, landing simultaneously on this middle vertex. This triggers the Exact Meet condition.

Odd Length ($L = 2k + 1$): P does not have a middle vertex, but rather a middle edge. At search depth k, s_F and s_B will have traversed k edges, landing on the respective endpoints of this middle edge. Because the graph G is undirected, the bridging edge connecting them exists in E, triggering the Adjacent Meet condition.

Thus, parity cannot cause undetected crossings. □

Corollary 1. *For state spaces with unit edge weights, the forward and backward search frontiers of BiXDFBnB intersect at a path length of k or $k+1$ from their respective origin nodes, guaranteeing that the searches meet in the middle.*

Proof. This follows directly from Lemmas 1 and 2. □

Theorem 1 (Completeness and Optimality)**.** *Given an admissible front-to-front heuristic h_{F2F}, BiXDFBnB is guaranteed to return a longest simple s-t path.*

Proof. By Lemmas 1 and 2, the algorithm generates only valid paths and will correctly detect the optimal path P^* (with maximum length L^*) if it is explored. It remains to show that prefixes of P^* are never prematurely pruned.

A branch is pruned if and only if $|s_F.\pi| + h_{F2F}(s_F, s_B) + |s_B.\pi| \leq |S|$. By definition, an admissible MAX heuristic guarantees that $h_{F2F}(s_F, s_B)$ is an upper bound on the length of any simple path connecting $s_F.head$ to $s_B.head$ using the remaining valid vertices. Therefore, the sum on the left side of the inequality represents an upper bound on the length of any path derivable from the current state pair. For P^* to be pruned, this upper bound must be $\leq |S|$, which implies $L^* \leq |S|$. This condition is only met if the incumbent solution S already has a length equal to or greater than that of P^*, guaranteeing that the search will ultimately return an optimal solution. □

4 Analysis of BiXDFBnB

4.1 Relation to SFBDS and Expansion Policies

Single-Frontier Bidirectional Search (SFBDS) (Felner et al. 2010) departs from traditional bidirectional search by using a single frontier instead of two. Each node in the SFBDS search tree can be seen as an independent task of finding

the shortest path between the current start and current goal. Likewise, our algorithm can be viewed as having a single frontier, where each node represents both a path from s and a path from t, with the task of finding the longest simple path that has the former as a prefix and the latter as a suffix.

One aim of SFBDS is to minimize search effort by choosing an appropriate *expansion policy* (AKA jumping policy), which decides which of two sides to expand next, choosing the direction with the highest potential for minimizing the total search effort. Three expansion policies are considered:

- **Alternating**: A simple baseline expansion policy that alternately expands the forward and the backward side. In domains with unit edge costs, like GLSP, this ensures that the search meets in the middle.
- **Greedy**: Performs a 1-step lookahead by peeking at all the children of x and measuring their heuristic towards y. Likewise, performs a 1-step lookahead by peeking at all the children of y to measure their heuristic towards x. Finally, it expands the side which contains the node with the largest heuristic value.
- **Simultaneous**: Performs a 2-step lookahead by peeking at all the children of x and all the children of y, and measuring their heuristic towards each other. Then, it expands the node that includes the child of x and the child of y with the largest heuristic value. This policy also ensures that the search meets in the middle in GLSP domains with unit edge costs.

Theoretical Trade-offs Each expansion policy balances node pruning against the computational overhead of heuristic evaluations, which scales with the branching factor (b). The **Alternating** policy is computationally cheap, requiring only b heuristic computations per expansion. While it guarantees balanced search depth, its rigid expansion order means it cannot leverage heuristic guidance to dynamically prioritize the more promising frontier. The **Greedy** policy improves pruning by independently evaluating the immediate successors of both frontiers, requiring $2b$ heuristic computations per expansion. This local perspective helps bypass poor branches, though evaluating each frontier in isolation means it cannot account for the direct heuristic interactions between opposing successors. The **Simultaneous** policy maximizes pruning power by evaluating the Cartesian product of all forward and backward successors, steering the search toward the best front-to-front connections. It requires b^2 heuristic computations per expansion—a quadratic overhead that risks outweighing the runtime gains from reduced node generation.

Empirical Comparison We conducted preliminary experiments testing each expansion policy on two domains: LSP in Grids and Coil-in-the-Box (CIB). Results, presented in Tables 1 and 2, show that the **Simultaneous** expansion policy usually outperforms the alternatives. Despite its quadratic per-expansion overhead, its superior pruning power drastically reduces overall node expansions. Across both domains, this reduction consistently outweighs the higher evaluation cost, yielding the best runtimes and solving capabilities in nearly all tested configurations.

Dim	Policy	20%		12%	
		Exp	Time(ms)	Exp	Time(ms)
6x6	Alternating	107	52	252	143
	Greedy	170	190	417	518
	Simultaneous	**52**	**36**	**132**	**97**
7x7	Alternating	167	133	2,694	1,973
	Greedy	347	544	5,557	8,601
	Simultaneous	**81**	**82**	**1,401**	**1,278**
8x8	Alternating	5,102	3,972	78,490	71,793
	Greedy	10,632	18,258	156,085	278,222
	Simultaneous	**2,707**	**2,470**	**39,373**	**44,301**

Table 1: Expansion Policies on LSP in square Grids

Dim	Policy	Finding		Proving	
		Exp	Time	Exp	Time
4D	Alternating	3	2	3	2
	Greedy	3	2	3	2
	Simultaneous	**2**	**1**	**2**	**1**
5D	Alternating	9	16	17	28
	Greedy	11	35	20	68
	Simultaneous	**5**	**11**	**9**	**26**
6D	Alternating	788	**2,462**	3,748	**14,998**
	Greedy	722	2,551	6,703	45,673
	Simultaneous	**512**	3,228	**2,190**	18,729
7D	Alternating	14,543	125,952	TO	TO
	Greedy	2,595,312	52,053,185	TO	TO
	Simultaneous	**3,265**	**56,960**	TO	TO

Table 2: Expansion Policies on Coil-in-the-Box

Given its expansions and runtime advantage, we adopt the **Simultaneous** expansion policy in BiXDFBnB experiments henceforth. Beyond the runtime benefits of massive node pruning, expanding both sides synchronously provides crucial algorithmic guarantees. As formally proven in Section 3.3 (Theoretical Properties), this synchronized topological growth rigorously ensures that the optimal path will be intercepted - either via an exact vertex meet or an adjacent edge meet - effectively bypassing the parity vulnerabilities common in traditional bidirectional search. Additionally, for state spaces with unit edge weights, the **Simultaneous** expansion policy guarantees that the forward and backward search frontiers meet in the middle. While our preliminary results show this policy dominates in GLSP and CIB, future work will explore whether these performance trends hold across a broader variety of domains and heuristic types.

4.2 F2F Heuristics in Bidirectional GLSP

In bidirectional search for GLSP, a search state is defined as a pair (s_F, s_B), where s_F is the forward path starting from s, and s_B is the backward path starting from t. The search progresses by extending either s_F or s_B. In this context, the *remaining graph* $G_r(G, s_F.\pi \cup s_B.\pi)$ is defined as G after removing (1) all vertices present in $s_F.tail \cup s_B.tail$, and (2) all other elements no longer allowed under the GLSP constraint L. Clearly, a paired state (s_F, s_B) is not a dead end only if $s_F.head$ and $s_B.head$ are in the same connected

component of $G_r(G, s_F.\pi \cup s_B.\pi)$. A F2F heuristic h is said to be *admissible* for bidirectional GLSP iff for every paired state (s_F, s_B) in the search space, $h(s_F, s_B)$ is *greater than or equal to* the weight of the longest constrained path (longest simple path in LSP) connecting $s_F.head$ to $s_B.head$ in $G_r(G, s_F.\pi \cup s_B.\pi)$. Crucially, because F2F heuristics evaluate the gap between the two frontiers, they can directly adapt existing F2E heuristics by treating the backward head, $s_B.head$, as if it were the fixed target t.

4.3 Examination of F2E vs. F2F

Here we examine the behavior of front-to-end (F2E) and front-to-front (F2F) heuristics within the context of BiXDFBnB. We argue that F2F is naturally suited to this paired-state algorithmic structure (akin to SFBDS), whereas F2E proves inadequate. We subsequently compare this bidirectional approach to unidirectional DFBnB, supporting our claims with the experimental results.

Front-to-Front Heuristics There is a major structural difference between our DFBnB algorithm and a standard best-first bidirectional algorithm with two distinct open lists. With two distinct open lists, the search independently grows two frontiers towards opposing goals. When frontiers intersect, a candidate path is formed (in GLSP we still need to perform a validation check, as detailed above). A F2F heuristic in that context acts as a *more-informed F2E heuristic* by calculating estimates between the current state and *all* states in the opposite frontier, selecting the best bound. While this provides stronger bounds than a F2E heuristic, the computational overhead of iterating over the entire opposing frontier typically outweighs the pruning benefits.

In contrast, the BiXDFBnB search operates on precisely one pair of states at a time, striving to bridge the gap between them. This affords F2F heuristics two decisive advantages. First, the distance calculation only needs to be executed between the two current heads, eliminating the quadratic algorithmic overhead. Second, the F2F heuristic calculates exactly what the paired search requires - a tight bound on the remaining path connecting the two specific states.

Front-to-End Heuristics In the context of our BiXDFBnB, using a F2E heuristic carries significant drawbacks. First, one must compute two independent heuristics (forward to t, backward to s) and aggregate them (taking the best bound), which is computationally wasteful. More fundamentally, these estimates ignore the constraints introduced by the opposing path blockages.

We can demonstrate the advantage of F2F heuristic over F2E heuristic in a DFBnB by looking at the remaining-graph heuristic (G_r). Suppose the forward and backward paths each have a g-value of 5. Using the F2E heuristic yields two independent bound calculations, where each remaining graph is constrained by only 5 visited vertices. Conversely, the true F2F heuristic evaluates a single remaining graph simultaneously constrained by all 10 visited vertices. Extracting a bound from a single, more severely depleted topology yields a consistently tighter admissible upper bound than taking the best of two less depleted topologies.

In unidirectional XDFBnB, only a single path is extended to the target, so the F2E and F2F heuristics converge by definition. For a mathematically fair comparison to a F2F heuristic where both paths have a length of 5, consider a unidirectional search that has reached depth 10. The remaining graph likewise drops 10 vertices. Despite matching vertex reduction, the bidirectional F2F heuristic frequently generates stronger bounds than the single-front heuristic. The bidirectional search isolates two distinct fragments of the environment, resulting in a significantly more fragmented remaining graph topology than a single, continuous 10-step path localized to one side. Given that heuristics like h_{BCC} heavily exploit graph fragmentation, F2F pruning becomes substantially harder to overcome.

The empirical evaluation (Section 5) confirms the theoretical advantages of F2F over F2E in BiXDFBnB, as well as BiXDFBnB's merits relative to unidirectional XDFBnB and other state-of-the-art approaches. Across both the LSP Grid and Maze domains, the BiXDFBnB F2F algorithm required orders of magnitude fewer node expansions and drastically lower wall-clock times compared to BiXDFBnB F2E. Additionally, BiXDFBnB F2F dominates XDFBnB in most metrics across all instances. This supports our reasoning that iteratively growing two opposing paths rather than extending a single long path breaks apart the remaining graph topology far more intensely, triggering earlier and more effective heuristic cutoffs.

5 Empirical Evaluation

Goals. Our evaluation addresses three main questions: (1) Does the use of a front-to-front (F2F) heuristic reduce node expansions compared to a front-to-end (F2E) heuristic in simultaneous bidirectional DFBnB? (2) Do reductions in node expansions translate into runtime improvements? (3) How does our approach compare to strong unidirectional and bidirectional baselines?

Domains. The evaluation includes the following domains:
(1) LSP in Grids. Here we find a longest simple path from s (top left corner) to t (bottom right corner) of a 2D grid. Grid sizes were all integer size combinations between 6×6 and 8×8 (a total of 6 combinations). For each size, we randomly set 8%, 12%, 16%, and 20% of the cells as obstacles, creating 10 random problem instances for each density (yielding 240 instances total). The heuristic used was h_{BCC}.
(2) Snake in Grids. In this domain, we search for the longest snake in grids spanning all integer size combinations from 7×7 to 9×9 (6 combinations). The random instance-generation method mirrored the one used for LSP in grids, yielding another 240 total instances. For this domain, a combination of the h_{BCC}, h_{SH}, and h_Y heuristics was utilized.
(3) LSP in Mazes. This domain shares the same setup as LSP in grids, except that the initial grid is the original maze used by Dahan et al. (2024), which serves as our single baseline instance (0 open diamonds). We generated increasingly difficult configurations by clearing randomly placed diamond-shaped areas within this base maze (Fig. 3 (left)). These areas are devoid of obstacles and create massive local state spaces. Specifically, we evaluated 10 random instances

with 1 open diamond and 10 random instances with 2 open diamonds, utilizing the h_{BCC} heuristic here as well.

(4) Coil-in-the-Box (CIB). Here, we find a coil-in-the-box simple cycle in a hypercube, with experiments spanning 4D, 5D, 6D, and 7D. Because the first four steps in SIB and CIB are forced due to symmetry, they can be safely assumed and dropped from the search. Consequently, we define our start and target vertices as $s = (0, \ldots, 1, 1, 1, 1)$ and $t = (0, \ldots, 0, 0, 0, 0)$. Heuristics were again a combination of h_{BCC}, h_{SH}, and h_Y.

Algorithms. Compared algorithms are: unidirectional A*, bidirectional best-first search (XMM), unidirectional depth-first branch-and-bound (XDFBnB), and BiXDFBnB with both front-to-end (F2E) and front-to-front (F2F) heuristics. As unidirectional search performance can be highly asymmetric in some graph topologies, we execute all unidirectional baselines in both forward ($s \rightarrow t$) and backward ($t \rightarrow s$) directions. Unless explicitly separated in data tables (as in the Maze domain), the reported values for unidirectional algorithms are the best performance out of the two directions, ensuring the strongest baseline comparison.

Experimental Setup. The evaluation was conducted using a 92-machine cluster, where each machine possesses an AMD EPYC 7763 CPU with 256 hyper-threads and 1000GB of RAM. Each combination of search algorithm and problem instance was allocated a single thread and a memory limit of 60GB. All algorithms were implemented in Python. This section highlights the main trends and representative results. Complete results, including both node expansions and runtimes for all instances appear in (Shubi et al. 2026). The code and data are publicly available (Shubi 2026).

5.1 Impact of Front-to-Front Heuristics

The central claim of this paper - that F2F heuristics are vastly superior to F2E heuristics in the context of simultaneous BiXDFBnB - is definitively validated across all domains. Table 3 presents a representative highlight of these results. As shown, **BiXDFBnB F2E** struggles significantly in environments with large open spaces, performing worse than or marginally comparable to traditional baselines. However, upgrading the heuristic to **F2F** reduces expansions by up to two orders of magnitude. Because the simultaneous expansion policy collapses the search to exactly one active pair of states, the quadratic overhead traditionally associated with F2F calculations is eliminated, translating this massive reduction in expansions directly into faster runtimes. The complete dataset comparing F2E and F2F across all tested domains and configurations appears in (Shubi et al. 2026).

5.2 Scalability in High-Dimensional CIB

The Coil-in-the-Box (CIB) problem - finding the longest constrained cycle in an n-dimensional hypercube - serves as a pure test of exponential scalability. We evaluate the algorithms on dimensions 4D through 7D.

A critical consideration in F2F heuristics is the trade-off between the time spent calculating a more informed heuristic and the time saved by expanding fewer nodes. Figure 1 illustrates this relationship. While the F2F calculation is slightly more expensive per node, the exponential reduction in the number of nodes expanded leads to a drastic improvement in overall wall-clock time. In the 7D hypercube, BiXDFBnB F2F finds a solution of optimal known length in only **56 seconds**, whereas the F2E variant and A* fail to complete within a 12-hour time limit. The full results for all CIB dimensions are provided in (Shubi et al. 2026).

Instance	Expansions		Runtime[ms]	
	F2E	F2F	F2E	F2F
Grid LSP 7×8 (12%)	165,241	**10,164**	370,323	**9,814**
Grid Snake 7×8 (12%)	7,430	**299**	59,729	**1,528**
Maze $od = 1$	515,170	**12,817**	2,461,587	**27,751**
Coil-in-the-Box 6D	163,561	**2,190**	1,375,600	**18,729**

Table 3: Comparison between F2E and F2F variants of BiXDFBnB in terms of node expansions and runtime.

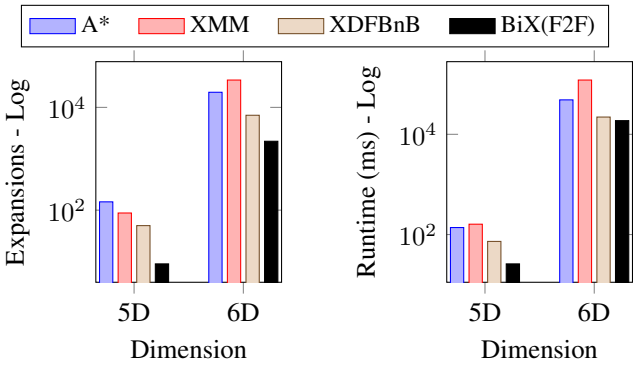


Figure 1: Coil-in-the-Box results. F2F's expansion reduction (left) directly enables its runtime advantage (right).

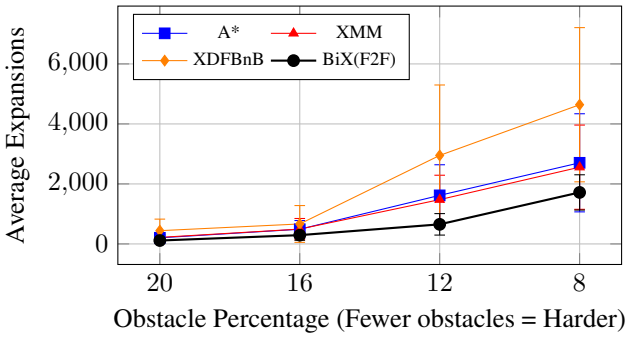
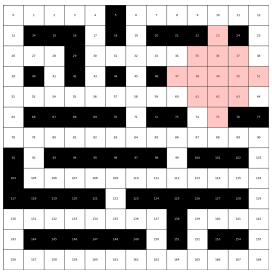


Figure 2: Average and standard deviation of expansions for Snake in 8×8 Grids with varying obstacle densities.

5.3 Performance on Grids and Mazes

For 2D environments, we evaluated both the standard Longest Simple Path (LSP) and the Snake problem (which strictly requires an induced path with no adjacent shortcuts) across square and rectangular grids seeded with varying percentages of obstacles ($20\%, 16\%, 12\%, 8\%$). As the number



Algorithm	0 Open Diamonds		1 Open Diamond		2 Open Diamonds	
	Expansions	Time[ms]	Expansions	Time[ms]	Expansions	Time[ms]
A* s→t	1,162	2,934	30,949	77,636	OOM	OOM
A* t→s	657	1,444	55,400	119,738	OOM	OOM
XMM	1,131	3,377	59,903	163,786	OOM	OOM
XDFBnB s→t	1,302	2,423	44,473	104,125	1,470,196	3,540,864
XDFBnB t→s	888	1,670	89,202	184,637	3,288,267	7,379,854
BiXDFBnB F2E	4,606	19,859	515,170	2,461,587	TO	TO
BiXDFBnB F2F	**501**	**948**	**12,817**	**27,751**	**187,759**	**433,233**

Figure 3: LSP in Mazes. Left: a maze with one cleared diamond. Right: results on the maze instances.

of obstacles decreases, the branching factor increases, making the state-space significantly harder to search.

LSP in Grids: Across most tested LSP grid instances, BiXDFBnB F2F substantially outperformed BiXDFBnB F2E and was often competitive with, or better than, the strongest unidirectional and best-first baselines. The complete tabular results for the LSP grid experiments can be found in (Shubi et al. 2026).

Snake in Grids: The added induced-path constraints of the Snake variant restrict the overall state space compared to pure LSP, but the relative performance gap between F2F and the baselines remains consistent across both problem types. Figure 2 illustrates the representative node expansions on 8×8 grids for the Snake problem. BiXDFBnB F2F dominates across all densities, leveraging its F2F pruning to significantly reduce the number of expansions.

LSP in Mazes: As shown in Fig. 3 (right), our proposed algorithm dominates all other algorithms on average in both expansions and runtime. The large state space in the 2 Open Diamonds instances caused the best-first algorithms A* and XMM to run out of memory (OOM) and BiXDFBnB F2E to time out (TO), while BiXDFBnB F2F outperforms even the best-performing unidirectional search, reducing average expansions by a factor of four and runtime by nearly an order of magnitude. To determine the statistical significance of the performance differences, we conducted a one-sided Wilcoxon signed-rank test ($\alpha = 0.05$). This non-parametric method can handle the heavily skewed distributions typical of search algorithm metrics. Across both the 1-Open and 2-Open Diamond configurations, the bidirectional algorithm yielded statistically significant improvements over both the $s \to t$ and $t \to s$ unidirectional variants in both total expansions and execution time. With all test pairings resulting in p-values below 0.005 (Wilcoxon test statistics $W \leq 3.0$), we confidently reject the null hypothesis, confirming that bidirectional search requires significantly fewer expansions and less runtime in these environments.

5.4 Summary of Results

Across all empirical domains, BiXDFBnB F2F emerges as the state-of-the-art solver for GLSP. By leveraging a depth-first branch-and-bound framework, it eliminates the exponential memory constraints of A* and XMM. Furthermore, by expanding nodes simultaneously, it mathematically guarantees path parity while unlocking the massive pruning power of front-to-front heuristics. This unique combination reduces node expansions by up to two orders of magnitude and translates directly into superior overall runtimes, rendering the traditional F2F computational overhead negligible.

6 Conclusion and Future Work

In this paper, we introduced BiXDFBnB, a novel bidirectional depth-first branch-and-bound algorithm designed for the Generalized Longest Simple Path (GLSP) problem. By evaluating opposing bounds simultaneously, our approach successfully integrates front-to-front (F2F) heuristics into a single-frontier search framework. While F2F heuristics are typically avoided in standard bidirectional search due to prohibitive computational overhead, our empirical evaluation demonstrates that in the context of GLSP - where state representation and path verification are inherently costly - F2F evaluations become not only viable but highly advantageous. BiXDFBnB effectively mitigates exponential memory bottlenecks that plague best-first algorithms such as A* and XMM, usually achieving reductions in both node expansions and overall runtime across Grids, Mazes, Snakes, and Coil-in-the-Box (CIB) domains.

Numerous directions exist for future work: **(1)** exploring additional, dynamic expansion policies. Our current implementation relies on a static, simultaneous policy to balance the forward and backward search frontiers. However, preliminary evaluations using k-step lookahead policies for $k \geq 2$ suggest that dynamically selecting the expansion direction based on maximum pruning potential can significantly reduce node expansions. Fully integrating and optimizing these dynamic lookahead policies - without incurring the prohibitive runtime overhead observed in higher dimensions - remains an open challenge. Importantly, this direction is also highly relevant to Single-Frontier Bidirectional Search (SFBDS) in minimization problems, providing a broader context to explore advanced expansion strategies. **(2)** Additionally, given the SFBDS tree we can try to search it with other algorithms rather than DFBnB, such as with A* or IDA*. **(3)** Finally, adapting the BiXDFBnB F2F framework to parallel processing architectures or extending it to other constrained maximization problems presents an exciting opportunity for further study.

Acknowledgments

Supported by the Israel Science Foundation (ISF) grant #909/23 and by the Frankel center for CS at BGU.

References

Chen, W. 2016. *The VLSI Handbook, Second Edition*, 77-31. Electrical Engineering Handbook. CRC Press. ISBN 9781420005967.

Cohen, Y.; Stern, R.; and Felner, A. 2020. Solving the Longest Simple Path Problem with Heuristic Search. In Beck, J. C.; Buffet, O.; Hoffmann, J.; Karpas, E.; and Sohrabi, S., eds., *ICAPS*, 75–79.

Dahan, G.; Tabib, I.; Shimony, S. E.; and Dinitz, Y. 2024. Generalized Longest Simple Path Problems: Speeding up Search Using SPQR Trees. In Felner, A.; and Li, J., eds., *Seventeenth International Symposium on Combinatorial Search, SOCS 2024, Kananaskis, Alberta, Canada, June 6-8, 2024*, 28–36. AAAI Press.

Dahan, G.; Tabib, I.; Shimony, S. E.; and Felner, A. 2022. Generalized Longest Path Problems. In *SoCS*, 56–64.

Felner, A.; Moldenhauer, C.; Sturtevant, N. R.; and Schaeffer, J. 2010. Single-Frontier Bidirectional Search. In Fox, M.; and Poole, D., eds., *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, 59–64. AAAI Press.

Holte, R.; Felner, A.; Sharon, G.; Sturtevant, N.; and Chen, J. 2017. MM: A bidirectional search algorithm that is guaranteed to meet in the middle. *Artif. Intell.*, 252: 232–266.

Kaindl, H.; and Kainz, G. 1997. Bidirectional Heuristic Search Reconsidered. *J. Artif. Intell. Res.*, 7: 283–317.

Karger, D.; Motwani, R.; and Ramkumar, G. D. 1997. On approximating the longest path in a graph. *Algorithmica*, 18(1): 82–98.

Kautz, W. H. 1958. Unit-Distance Error-Checking Codes. *j-IRE-TRANS-ELEC-COMPUT*, EC-7(2): 179–180.

Palombo, A.; Stern, R.; Puzis, R.; Felner, A.; Kiesel, S.; and Ruml, W. 2015. Solving the Snake in the Box Problem with Heuristic Search: First Results. In *Symposium on Combinatorial Search (SoCS)*, 96–104.

Portugal, D.; and Rocha, R. 2010. MSP Algorithm: Multi-robot Patrolling Based on Territory Allocation Using Balanced Graph Partitioning. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, 1271–1276. New York, NY, USA: ACM. ISBN 978-1-60558-639-7.

Schmidt, K.; and Schmidt, E. G. 2010. A Longest-Path Problem for Evaluating the Worst-Case Packet Delay of Switched Ethernet. In *SIES*, 205–208. IEEE.

Shubi, T. 2026. Implementation. https://github.com/tzurshubi/BiHS/releases/tag/paper-v2.0.

Shubi, T.; Felner, A.; Shimony, S. E.; and Shperberg, S. S. 2026. Bidirectional Search for Longest Paths: Case for Front-to-Front Heuristics (Extended Version). *arXiv preprint arXiv:2606.05956*. Available at https://arxiv.org/abs/2606.05956.

Shubi, T.; Shimony, S. E.; Felner, A.; and Shperberg, S. S. 2025. Bidirectional Heuristic Search in Longest Path Problems. In Lynce, I.; Murano, N.; Vallati, M.; Villata, S.; Chesani, F.; Milano, M.; Omicini, A.; and Dastani, M., eds., *ECAI 2025 - 28th European Conference on Artificial Intelligence, 25-30 October 2025*, volume 413, 4840–4847. IOS Press.

Stern, R. T.; Kiesel, S.; Puzis, R.; Felner, A.; and Ruml, W. 2014. Max is More than Min: Solving Maximization Problems with Heuristic Search. In *SOCS*, 148–156. AAAI Press.

Wong, W. Y.; Lau, T. P.; and King, I. 2005. Information Retrieval in P2P Networks Using Genetic Algorithm. In *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, WWW '05, 922–923. New York, NY, USA: ACM. ISBN 1-59593-051-5.