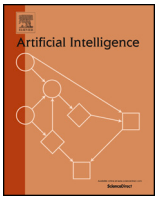
Artificial Intelligence 348 (2025) 104420

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Bridging theory and practice in bidirectional heuristic search with front-to-end consistent heuristics

Lior Siag [*], Shahaf S. Shperberg [*]

Faculty of Computer and Information Science, Ben-Gurion University of the Negev, 1 David Ben-Gurion Blvd., Be'er Sheva, 8410501, Israel

ARTICLE INFO

Keywords:
Heuristic search
Consistent heuristics
Bidirectional search

ABSTRACT

Recent research on bidirectional heuristic search (BiHS) has been shaped by the *must-expand pairs* (MEP) theory, which identifies the pairs of nodes that must be expanded to ensure solution optimality. Another line of research has focused on algorithms utilizing lower bounds derived from consistent heuristics during the search. This paper bridges these two approaches, offering a unified framework that demonstrates how both existing and novel algorithms can be derived from MEP theory. We introduce an extended set of bounds, encompassing both previously known and newly formulated ones. Using these bounds, we develop a range of algorithms, each employing different criteria for termination, node selection, and search direction. Finally, we empirically evaluate how these bounds and algorithms impact search efficiency.

1. Introduction

Heuristic search is a key aspect of Artificial Intelligence, enabling efficient problem-solving across various domains. Bidirectional Heuristic search (BiHS), in particular, enhances search efficiency by simultaneously exploring paths from both the start and goal states, often reducing the search space and time required to find optimal solutions. Research on BiHS dates back to the late 1960s. More recently, the theory of *must-expand pairs* (MEP) was developed [7], laying the foundation for a series of important advances in the field [19,20,22,21,27,2,1]. MEP theory characterizes the forward and backward node pairs that must be expanded to guarantee optimality. Recent BiHS algorithms, such as NBS [4] and DVCBS [23], leverage this theory to minimize search effort while returning optimal solutions.

This paper focuses on the scenario where the heuristic is consistent. In such cases, the conditions for determining which pairs of nodes must be expanded can be tightened [20], reducing the number of MEPs. These refined conditions, based on heuristic consistency, offer the potential to further reduce the number of expanded nodes. However, algorithms like NBS and DVCBS cannot efficiently incorporate these tighter conditions, as doing so requires significantly more computational effort.

Algorithms specifically designed for consistent heuristics have emerged independently of the MEP theory. These approaches utilize information derived from heuristic consistency to maintain lower bounds (termed *search bounds*) on the solution cost, thereby accelerating the search [11,17,2,18]. Key questions include whether additional search bounds can be developed and how effectively they might perform.

* Corresponding author.
E-mail addresses: siagl@post.bgu.ac.il (L. Siag), shperbsh@bgu.ac.il (S.S. Shperberg).

https://doi.org/10.1016/j.artint.2025.104420
Received 14 October 2024; Received in revised form 15 September 2025; Accepted 15 September 2025

Available online 19 September 2025

To address these questions, it is essential to bridge the conceptual gap between MEP theory and the algorithms based on search bounds. This paper provides a unifying perspective that closes this gap, making several key contributions: [1]

1. We introduce a method for deriving search bounds directly from MEP conditions. This method can reproduce all existing search bounds and generate novel ones, with the potential to produce infinitely many new bounds. We rigorously prove that all derived bounds are valid lower bounds on the optimal solution during search. Additionally, we propose a method for identifying effective bounds by solving a small set of instances in each domain and generalizing to other instances.

2. We present a comprehensive algorithmic framework that exploits any subset of bounds, with an emphasis on using individual bounds to guide the search. This framework generalizes and encompasses existing search-bound-based algorithms.

3. We propose new policies for node selection and search direction, yielding different algorithmic variants within the framework, each tailored to optimize search efficiency.

4. We conduct an extensive experimental evaluation, analyzing the impact of each bound, node-selection policy, and direction-selection policy. We compare the node expansions of these variants against the theoretical lower bound from MEP theory, demonstrating that existing algorithms closely approach theoretical efficiency limits.

2. Definitions, notations, and background

In BiHS, the aim is to find a least-cost path, of cost C^*, between two states *start* and *goal* in a given graph G. $dist(x, y)$ denotes the shortest distance between x and y, so $dist(start, goal) = C^*$. In some cases, the cost of the cheapest edge of the graph (denoted ϵ) is known; otherwise, ϵ is assumed to be 0.

BiHS typically keeps two open lists, OPEN_F for the forward search (F), and OPEN_B for the backward search (B). Given a direction D (either F or B), we use f_D, g_D and h_D to indicate f-, g-, and h-values in direction D. We use $\overline{D}$ to denote the direction that is opposite to D, and define $f_{\overline{D}}$, $g_{\overline{D}}$ and $h_{\overline{D}}$ symmetrically. This paper considers the two *front-to-end* heuristic functions [11], $h_F(s)$ and $h_B(s)$ which respectively estimate $dist(s, goal)$ and $dist(start, s)$ for all $s \in G$. h_F is *forward admissible* iff $h_F(s) \leq dist(s, goal)$ for all s in G and is *forward consistent* iff $h_F(s) \leq dist(s, s') + h_F(s')$ for all s and s' in G. Backward *admissibility* and *consistency* are defined analogously for h_B.

In addition to the known g, h, and f functions, we define the following functions which are used throughout the paper:

1. $d_D(n) = g_D(n) - h_{\overline{D}}(n)$, the *difference* between the actual cost of node n in direction D and its heuristic estimate; this indicates the *heuristic error* for node n.
2. $b_D(n) = f_D(n) + d_D(n)$. $b_D(n)$ adds the heuristic error $d_D(n)$ to $f_D(n)$ to indicate that the opposite search using $h_{\overline{D}}(n)$ will underestimate by $d_D(n)$.
3. $rf_D(n) = g_D(n) - h_D(n)$, the reverse f-value [1], a function that is similar to f, but which subtracts the heuristic instead of adding it.
4. $rd_D(n) = g_D(n) + h_{\overline{D}}(n)$, the reverse d-value [1], which adds the heuristic estimation towards the opposite direction instead of subtracting it. The term *reverse* for rf and rd was coined by Alcázar [1]. Although the motivation behind rf and rd may seem unintuitive, they can be used as building blocks for some of the bounds on optimal solutions' cost, as we show below.

Finally, let $xMin_D$ denote the minimal x value in OPEN_D. For example, $gMin_D = \min_{n \in \text{OPEN}_\text{D}} g_D(n)$, the minimal g-value in direction D. Additionally, $\{x, y\}Min_D$ is the minimal $x + y$ value in OPEN_D, e.g., $\{rf, d\}Min_D = \min_{n \in \text{OPEN}_\text{D}} \{rf_D(n) + d_D(n)\}$. Likewise, $\{x, y, z\}Min_D$ is the minimal $x + y + z$ value in direction D.

2.1. Must-expand nodes in bidirectional search

Any unidirectional heuristic search (UniHS) algorithm (meeting reasonable theoretical assumptions), that is guaranteed to find an optimal solution on every problem with an admissible heuristic, *must expand* all nodes n with $f(n) < C^*$ to prove the optimality of solutions when given a consistent heuristic [5].

The generalization of this theory to BiHS [7] showed that in BiHS the *must-expand* property is defined on *pairs* of nodes (u, v) from the forward and backward frontiers (and not on a single node as in UniHS) as follows:

$$lb(u, v) = \max\{f_F(u), f_B(v), g_F(u) + g_B(v) + \epsilon\} \tag{1}$$

[1] A portion of this research was previously published [25]. This paper builds on those earlier results in several directions. First, we introduce linear-combination-based bounds, which generalize the framework to encompass infinitely many possible bounds. We prove that any convex combination constitutes a valid lower bound and provide methods to efficiently select weights for the linear combination. On the empirical side, we present a substantially more comprehensive evaluation, including analyses of termination conditions, direction-selection strategies, and linear-combination bounds. In addition, we offer an explanation and example of MVC calculation based on Koenig's theorem, and, more broadly, present more rigorous derivations and detailed analyses throughout.

$lb(u,v)$ is a lower bound on the cost of any path that can connect *start* and *goal* via u and v. In BiHS, a pair of nodes (u,v) is called a *must-expand pair* (denoted MEP) if $lb(u,v) < C^*$. In a MEP at least one of u or v *must be expanded*. Otherwise, an algorithm that does not expand either u or v when $lb(u,v) < C^*$ might miss the optimal solution.

The set of MEPs can be reformulated as a bipartite graph, denoted as the *Must-Expand Graph* (GMX) [4]. For each state $u \in G$, GMX includes a forward vertex u_F and a backward vertex u_B. For each pair of states $u, v \in G$, there is an edge in GMX between u_F and v_B iff (u,v) is a MEP. Since each edge of GMX is a MEP and at least one node from each MEP must be expanded to ensure the optimality of the solution, it follows that any optimal algorithm must expand a *vertex cover* of GMX. Thus, the minimum number of node expansions required to guarantee the optimality of a solution for a problem instance by BiHS algorithms is the size of the *minimum vertex cover* (denoted by MVC) of GMX.

2.2. MEPs when assuming a consistent heuristic

This paper focuses on the case where algorithms assume a consistent heuristic (i.e., they do not need to return an optimal solution when given admissible heuristics that are not consistent). We denote this as the *consistency case*, under which more information can be exploited, and the aim is to develop algorithms that leverage this assumption. For the consistency case, Shaham et al. [20] introduced tighter lower bounds for pairs denoted $lb_C(u,v)$ (C for consistency):

$$lb_C(u,v) = g_F(u) + g_B(v) + \max \begin{cases} h_F(u) - h_F(v) & \text{(2a)} \\ h_B(v) - h_B(u) & \text{(2b)} \\ \epsilon & \text{(2c)} \end{cases}$$

All terms in the max expression are lower bounds on $dist(u,v)$ and can thus be added to $g_F(u) + g_B(v)$. ϵ (term 2c) is a trivial lower bound, term 2a is derived from the definition of a forward-consistent heuristic: $h_F(u) \leq dist(u,v) + h_F(v) \implies h_F(u) - h_F(v) \leq dist(u,v)$, and term 2b is derived analogously from the definition of backward consistency.[2] $lb_C(u,v)$ induces a new MEP definition, denoted as MEP_C. Since $lb_C(u,v) \geq lb(u,v)$ for every pair of nodes u, v, the number of MEP_Cs is *smaller than or equal to* the number of MEPs for any given problem instance.

Finally, for undirected graphs, $lb_C(u,v)$ can be further tightened resulting in $lb_{CU}(u,v)$ (U for undirected):

$$lb_{CU}(u,v) = g_F(u) + g_B(v) + \max \begin{cases} h_F(u) - h_F(v) & \text{(3a)} \\ h_B(v) - h_B(u) & \text{(3b)} \\ \epsilon & \text{(3c)} \\ h_F(v) - h_F(u) & \text{(3d)} \\ h_B(u) - h_B(v) & \text{(3e)} \end{cases}$$

Both Equations (2) and (3) can be used for defining corresponding GMX graphs.

2.3. NBS and DVCBS

Near-Optimal Bidirectional Search (NBS) [4] is a prominent algorithm derived from the MEP theory. At every expansion cycle, NBS chooses a pair of nodes u, v with a minimal $lb(u,v)$ (denoted as LB) among all pairs in the open lists, and expands *both* nodes. This approach is based on the 2-factor approximation of minimal vertex covers [15]. Thus, NBS is guaranteed to expand a vertex cover of GMX whose size is at most $2 \times |MVC|$.

Obtaining a pair of nodes u, v from all possible combinations such that the lower bound they induce satisfies $lb(u,v) \leq LB$, or determining that no such pair exists and increasing LB accordingly, can be implemented naively by inserting all node pairs into an open list. Each pair is prioritized by its $lb(u,v)$ value, and the pair with the smallest priority is selected at each iteration. However, this approach incurs a time complexity of $O(n^2 \log n^2)$, where n is the number of frontier nodes, due to the need to represent and manage all possible node pairs. In contrast, typical search algorithms operate with a time complexity of only $O(n \log n)$, rendering the naive approach impractical.

Fortunately, the structure of the lb function (Equation (1)) allows for a more efficient computation that is only linear in the number of nodes [4]. Specifically, the first two terms of the max expression in Equation (1) depend only on $f_F(u)$ and $f_B(v)$, respectively. Thus, nodes whose f-values exceed LB can be filtered locally in each direction. This leaves only the third condition, $g_F(u) + g_B(v) + \epsilon \leq LB$, which can then be efficiently checked by sorting the filtered nodes in each direction by non-decreasing g-value, and identifying the pair (u,v) with minimal combined g-value.

[2] Equation (1), which defines $lb(u,v)$, includes terms for $f_F(u)$ and $f_B(v)$. These terms are redundant in $lb_C(u,v)$, as $g_F(u)+g_B(v)+h_F(u)-h_F(v) = f_F(u)+d_B(v) \geq f_F(u)$, and for $f_B(v)$ using term 2b.

To support this process, NBS employs two-level priority queues in each direction: a *waiting queue*, which stores all nodes n with $f_D(n) > LB$, sorted by f-value, and a *ready queue*, which stores all nodes n with $f_D(n) \leq LB$, sorted by g-value. These data structures enable NBS to only expand MEPs, while maintaining an amortized insertion and deletion cost of $\log(n)$, where n is the number of frontier nodes.

Dynamic Vertex Cover Bidirectional Search (DVCBS) [23] is an alternative to NBS. DVCBS constructs a partial, dynamic GMX (denoted DGMX) based on the nodes currently in OPEN, finds an MVC of DGMX, and expands all nodes in this MVC. DVCBS is unbounded in the worst case, but empirically outperforms NBS on many benchmarks.

2.3.1. Impracticality of utilizing lb_C or lb_{CU} directly

A fundamental challenge arises when adapting NBS and DVCBS to the consistent heuristic setting. Specifically, it becomes challenging to efficiently obtain a pair of nodes u, v such that the lower bound they induce satisfies $lb(u, v) \leq LB$, or to determine that no such pair exists and increase LB accordingly. Unlike the original definition of lb in Equation (1), where only one term in the max expression depends jointly on u and v, in the consistent variants—lb_C (Equation (2)) and lb_{CU} (Equation (3))—multiple terms depend on both nodes. As a result, the two-level priority queue used in NBS is no longer suitable for efficiently identifying a pair that minimizes lb_C or lb_{CU}. A similar limitation applies to DVCBS.

One might consider maintaining separate open lists for finding a pair that minimizes each term in lb_C and selecting the maximum of their minimal values as a candidate lower bound. However, this strategy can be misleading. Consider the following example. Let the forward nodes be $u_1 = (10, 10, 10)$, $u_2 = (10, 0, 0)$, and $u_3 = (10, 4, 2)$, where each tuple denotes (g_F, h_F, h_B). Similarly, let the backward nodes be $v_1 = (10, 10, 10)$, $v_2 = (10, 0, 0)$, and $v_3 = (10, 2, 4)$. Assume $\epsilon = 0$, so the third term in the max expression of lb_C is always dominated by the first two and thus can be ignored. Selecting a pair that minimizes the first term would yield (u_2, v_1), resulting in

$$g_F(u_2) + g_B(v_1) + h_F(u_2) - h_F(v_1) = 10.$$

Similarly, minimizing the second term gives (u_1, v_2) with

$$g_F(u_1) + g_B(v_2) + h_B(v_2) - h_B(u_1) = 10.$$

Taking the maximum of these two estimates suggests a bound of 10. However, in both cases, $lb_C(u_2, v_1) = 30$ (due to the second term) and $lb_C(u_1, v_2) = 30$ (due to the first term). The true minimum is achieved by the pair (u_3, v_3), for which $lb_C(u_3, v_3) = 12$, yielding a significantly better bound.

To the best of our knowledge, there exists no known data structure capable of returning a pair with minimal lb_C (or lb_{CU}) without incurring a quadratic computational overhead in the number of frontier nodes (i.e., considering all possible pairs).

In an effort to develop a somewhat efficient algorithm for the consistency case, Alcázar [1] introduced a method called bucket-to-bucket (BTB). This scheme involves grouping nodes into buckets that share the same $g_D, h_D, h_{\overline{D}}$ values. Using these buckets, both NBS and DVCBS can be applied using lb_C (or lb_{CU}). However, each expansion cycle of a bucket requires a computational overhead that is quadratic in the number of buckets. While the number of buckets may be small in certain domains, in general, there could be an arbitrary number of buckets. As a result, the quadratic runtime per expansion cycle may become impractical (e.g., in road networks or grids).

2.4. Search bounds, DIBBS/BAE*, and DBBS

We use the term *search bounds* [2] to denote all lower bounds on the solution cost that can be computed during the search. We classify the search bounds into three categories. **(1)** *Global bounds* provide a lower bound on the cost of every solution from *start* to *goal*. These bounds can be used as termination conditions; when the incumbent solution has a cost that equals the bound then it is optimal. **(2)** *Individual-node bounds* bound the cost of any solution from *start* to *goal* that passes through a given node n (e.g. $lb(n)$). **(3)** *Pair bounds* bound the cost of paths from *start* to *goal* that passes through a given pair of nodes u, v, where $u \in \text{OPEN}_\text{F}$ and $v \in \text{OPEN}_\text{B}$ (e.g., $lb(u, v)$).

Several search bounds have been used in the heuristic search literature. In UniHS the minimal f-value in the open list, $fMin$, is a global bound on the cost of any solution. This f-*bound* is commonly used in the termination conditions of many algorithms (both unidirectional and bidirectional). Similarly, in BiHS, $fMin_F$ and $fMin_B$ are global bounds. Another bound in BiHS is the g-*bound* $= gMin_F + gMin_B + \epsilon$.

For the consistency case, other search bounds have been proposed. Kaindl and Kainz [11] proposed adding heuristic errors (defined as $d_D(n)$ above), of direction D, to f-values of the opposite direction $\overline{D}$. In particular, they defined the following individual-node bounds $KKAdd_D(n) = f_D(n) + dMin_{\overline{D}}$ and symmetrically, $KKMax_D(n) = d_D(n) + fMin_{\overline{D}}$. Note that these node bounds can be transformed to be global bounds by taking the minimal f- and d-values from both directions as follows: $KKAdd = fMin_D + dMin_{\overline{D}}$ and $KKMax = dMin_D + fMin_{\overline{D}}$. A lower bound B_i is said to *dominate* another lower bound B_j iff $B_i \geq B_j$. Since $fMin_D + dMin_{\overline{D}} \geq fMin_F$ then $KKAdd$ dominates $fMin_F$. Similarly, $KKMax$ dominates $fMin_B$. Another global bound is the foundation of two identical algorithms, BAE* [17] and DIBBS [18], which expand nodes with minimal b-value. They introduce a new global bound, b-*bound* $= (bMin_F + bMin_B)/2$, and terminate when a solution is found with cost equal to the b-bound.

Alcázar [1] combined the KK bounds, the g-bound, and the b-bound together to improve their individual performance. In this context, a node n is defined as *expandable* iff:

$$n \in \underset{n' \in \text{OPEN}_D}{\text{argmin}} (\max\{f_D(n') + dMin_{\overline{D}}, d_D(n') + fMin_{\overline{D}}, \tag{4}$$

$$(b_D(n') + bMin_{\overline{D}})/2, \tag{5}$$

$$g_D(u) + gMin_{\overline{D}} + \epsilon\}) \tag{6}$$

For an undirected graph, two more bounds were introduced: $rfMin_F + rdMin_B$ (forward rc, meaning reverse consistent) and $rdMin_F + rfMin_B$ (backward rc). The minimal values in the open lists, which are used for defining the global search bounds, can be computed with respect to the set of expandable nodes. This creates a *fixed-point computation* in which the minimal values in the open lists are updated based on the set of expandable nodes, and the set of expandable nodes is updated based on the updated minimal values in the open lists until convergence is reached. The DBBS algorithm [1] performs this fixed-point computation to find tighter bounds, and thus can possibly expand fewer nodes during the search. However, this fixed-point computation is computationally expensive in practice.

3. Deriving bounds from the MEP theory

In this section, we introduce a unifying view for the consistency case and draw a connection between lb_C (Equation (2)) and all existing search bounds (presented in Section 2.4). In addition, we show that additional search bounds can be derived from lb_C and even more bounds can be derived when also considering undirected graphs (lb_{CU}, Equation (3)).

A first step towards a unifying view was taken by Alcázar [1] who observed that taking each element in the max term of Equation (2) (2a, 2b, and 2c) individually, can derive the *KKAdd*, *KKMax*, and g-bound, correspondingly. For example, for term 2a, $g_F(u) + g_B(v) + h_F(u) - h_F(v) = f_F(u) + d_B(v) \geq fMin_F + dMin_B = KKAdd$. However, the method of taking individual elements of the max term is limited and cannot be used for producing other bounds, e.g., the b-bound. We now present a general method to produce additional bounds from the max term. In fact, any max function over a set of elements S can be bounded from below as follows:

$$\max S \geq \frac{\sum_{s' \in S'} s'}{|S'|} \qquad \forall S' \in \mathcal{P}(S) \setminus \emptyset \tag{7}$$

Here $P(S)$ denotes the power set of S. The average of the elements of any non-empty subset of S is always a lower bound on the maximum element of S. Using this rule, the max expression in Equation (2) can be lower-bounded by different subsets S' of the terms inside (i.e., 2a, 2b, and 2c). For example, the derivation using $S' = \{2a, 2b\}$ results in the b-bound (B_4 below):

$$\begin{aligned} lb_C(u,v) &\geq g_F(u) + g_B(v) + \frac{[\text{term } 2a] + [\text{term } 2b]}{2} \\ &= \tfrac{2g_F(u)+2g_B(v)+h_F(u)-h_F(v)+h_B(v)-h_B(u)}{2} \\ &= 1/2 \cdot (f_F(u) + d_F(u) + f_B(v) + d_B(v)) \\ &= 1/2 \cdot (b_F(u) + b_B(v)) \geq 1/2 \cdot (bMin_F + bMin_B) \\ &= b\text{-bound } (B_4) \end{aligned}$$

By considering all subsets S' of terms 2a, 2b, and 2c, and for a node n by replacing the value $x_D(n)$ with the minimal x-value in OPEN_D, $xMin_D$, as was demonstrated above (for $2a$ and for $2a, 2b$), we get the following global bounds (recall that notation, such as $\{f,g\}Min_F$, are as defined in Section 2):

B_1: $fMin_F + dMin_B$ (*KKAdd*), when $S' = \{2a\}$
B_2: $dMin_F + fMin_B$ (*KKMax*), when $S' = \{2b\}$
B_3: $gMin_F + gMin_B + \epsilon$ (g-bound), when $S' = \{2c\}$
B_4: $\frac{bMin_F + bMin_B}{2}$ (b-bound), when $S' = \{2a, 2b\}$
B_5: $\frac{\{f,g\}Min_F + \{d,g\}Min_B + \epsilon}{2}$, when $S' = \{2a, 2c\}$
B_6: $\frac{\{d,g\}Min_F + \{f,g\}Min_B + \epsilon}{2}$, when $S' = \{2b, 2c\}$
B_7: $\frac{\{b,g\}Min_F + \{b,g\}Min_B + \epsilon}{3}$, when $S' = \{2a, 2b, 2c\}$

The derivation of each of the bounds $\{B_1, \ldots, B_7\}$ can be done similarly to the two examples presented above. The full derivation is detailed in Appendix A. While each term in Equation (2) can be used to track a different bound, yielding B_1, B_2, and B_3, respectively, there are situations where the other bounds provide a tighter lower bound on C^*. Numerical examples demonstrating that, for each bound B_i (from B_1 to B_7), there exists a case in which B_i strictly dominates all other bounds—i.e., yields a higher estimate of the solution cost—are provided in Appendix B. Thus, none of the bounds dominate the others in all cases.

We note that substantial work was done to individually prove each of the previously proposed bounds (e.g., Kaindl and Kainz [11], Sewell and Jacobson [18]), whereas the derivation of these bounds from the MEP theory is straightforward. Finally, note that individual-node bounds that correspond to each of the above global bounds $\{B_1, \ldots, B_7\}$ can be derived from the MEP theory as well. This is done by replacing the value of $x_D(n)$ with the minimal x value in OPEN_D ($xMin_D$) in only one of the two

directions (rather than in both), as in the derivations above. For example, the last step in the *KKAdd* derivation above (B_1) becomes $f_F(u) + d_B(v) \geq f_F(u) + dMin_B$.

3.1. Bounds for undirected graphs

For undirected graphs, we derive lower bounds from lb_{CU} instead of lb_C. Since terms 2a, 2b, and 2c are identical to 3a, 3b, and 3c, the above seven bounds $\{B_1, \ldots, B_7\}$ are also valid for lb_{CU}. Nevertheless, additional bounds for the max expression can be derived from terms 3d and 3e. At first glance, one might count 31 possible non-empty subsets of terms 3a–3e, suggesting 31 candidate bounds. However, note that terms 3a and 3d cancel each other ($3a + 3d = h_F(u) - h_F(v) + h_F(v) - h_F(u) = 0$), so any subset containing both is dominated by the subset that omits one of them. The same holds for 3b and 3e; hence, only 17 non-dominated bounds remain. The resulting (additional) bounds and their respective subsets of terms are as follows:

B_8: $rfMin_F + rdMin_B$ (*forward rc*), when $S' = \{3d\}$
B_9: $rdMin_F + rfMin_B$ (*backward rc*), when $S' = \{3e\}$
B_{10}: $\frac{\{f,rd\}Min_F + \{rf,d\}Min_B}{2}$, when $S' = \{3a, 3e\}$
B_{11}: $\frac{\{rf,d\}Min_F + \{f,rd\}Min_B}{2}$, when $S' = \{3b, 3d\}$
B_{12}: $\frac{\{rf,g\}Min_F + \{rd,g\}Min_B + \epsilon}{2}$, when $S' = \{3c, 3d\}$
B_{13}: $\frac{\{rd,g\}Min_F + \{rf,g\}Min_B + \epsilon}{2}$, when $S' = \{3c, 3e\}$
B_{14}: $\frac{\{rf,rd\}Min_F + \{rf,rd\}Min_B}{2}$, when $S' = \{3d, 3e\}$
B_{15}: $\frac{\{f,rd,g\}Min_F + \{rf,d,g\}Min_B + \epsilon}{3}$, when $S' = \{3a, 3c, 3e\}$
B_{16}: $\frac{\{rf,d,g\}Min_F + \{f,rd,g\}Min_B + \epsilon}{3}$, when $S' = \{3b, 3c, 3d\}$
B_{17}: $\frac{\{rf,rd,g\}Min_F + \{rf,rd,g\}Min_B + \epsilon}{3}$, when $S' = \{3c, 3d, 3e\}$

The above set of bounds ($\{B_1, \ldots, B_7\}$ and $\{B_1, \ldots, B_{17}\}$ for Equation (2) and Equation (3), respectively) are all the possible bounds that can be produced when approximating the max term in the equations using Equation (7). Other lower bounds would require bounding the max terms in different ways or using other methods altogether. Another possibility for generating new bounds is to introduce additional assumptions. For instance, as shown by Alcázar et al. [2], if the greatest common denominator among the cost of all non-zero-cost edges, denoted as ι, is known, it can be used to tighten the bounds further. For each bound B_i from the above set of bounds, we can use $\iota\lceil \frac{B_i}{\iota} \rceil$ as a possibly tighter version of B_i. So, in unit-edge-cost graphs, all of the above bounds can be rounded up.

3.2. Convex-combination approximation of the max expression

So far, we have established existing bounds from the literature and introduced novel ones by approximating the maximum expression as an average of a subset of its elements, as elaborated in Equation (7). However, this approximation can be generalized as a special case of taking a convex combination of the terms in the maximum expression.

Formally, a convex combination of points $x_1, x_2, \ldots, x_n$ in a vector space is a linear combination of these points where the coefficients are non-negative and sum up to 1. Mathematically, it can be expressed as:

$$\sum_{i=1}^{n} \lambda_i x_i$$

where $\lambda_i \geq 0$ for all i and $\sum_{i=1}^{n} \lambda_i = 1$.

We show that every convex combination of the terms S is a lower bound of $\max S$.

Theorem 1. *Every convex combination of terms within a maximum expression serves as a lower bound for that expression.*

Proof. Given a maximum expression of $S = s_1, \ldots, s_n$ terms, $\max_{s \in S} s$, let $\lambda = [\lambda_1, \ldots, \lambda_n]$ represent the coefficient of some convex combination of the terms in S. Let $s^* = \text{argmax}_{s \in S}$ be the term that achieves the maximum value in the max expression.

The convex combination represented by λ can be written as:

$$\sum_{i=1}^{n} \lambda_i s_i$$

Since $s_i \leq s^*$ and $\lambda_i \geq 0$ for all i,

$$\sum_{i=1}^{n} \lambda_i s_i \leq \sum_{i=1}^{n} \lambda_i s^* = (\sum_{i=1}^{n} \lambda_i) \cdot s^*$$

Since λ are the coefficients of a convex combination, $(\sum_{i=1}^{n} \lambda_i) = 1$, thus, $\sum_{i=1}^{n} \lambda_i s_i \leq s^*$. □

Algorithm 1 BiHS General Algorithmic Framework.

```
U ← ∞, LB ← ComputeLowerBound()
while OPEN_F ≠ ∅ ∧ OPEN_B ≠ ∅ ∧ U > LB do
    D ← ChooseDirection()
    n ← ChooseNode(D)
    Expand(n, D)                                   ▷ update U
    LB ← ComputeLowerBound()
return U
```

Consequently, there are infinitely many possible bounds for every max expression. Notably, the bounds derived from Equation (7) are a special case of convex combination. Specifically, given a maximum expression $\max S$, where $S = s_1, \ldots, s_n$, and a subset $S' \subseteq S$, a convex combination $\lambda = [\lambda_1, \ldots, \lambda_n]$ can be defined, as follows:

$$\lambda_i = \begin{cases} \frac{1}{|S'|} & \text{if } s_i \in S' \\ 0 & \text{otherwise} \end{cases}$$

Thus, treating the bounds as a linear combination of terms generalizes upon our previous findings. However, since considering infinitely many bounds is infeasible, we later propose a method for identifying effective bounds.

4. Using the bounds within algorithms

We now turn to the practical aspects of the proposed bounds. We describe a general BiHS algorithmic framework for the consistency case, presented in Algorithm 1, and show how to utilize information from the bounds in this framework. There are three decisions that define BiHS algorithms: (1) deciding when to terminate the search, (2) choosing which node to expand from a given direction, and (3) choosing the direction from which the next node will be expanded. Each of these can utilize information from the bounds.

4.1. Termination condition

Each global bound B_i can be used to prove that the incumbent solution is optimal, i.e., when a solution is found with $cost = B_i$, the algorithm can terminate. Naturally, when there are several lower bounds, their maximum is the tightest lower bound among them and can be used as a termination condition. The advantage of using several bounds is that if we have the optimal solution U (with cost C^*) in hand, we can halt faster: once $B_i = C^*$ for any of the available global bounds B_i. The tradeoff is the overhead of maintaining and consulting several bounds. In addition, the bounds can be computed either with the (computationally expensive) fixed-point computation, as in DBBS, or without it.

4.2. Choosing which node to expand

The decision of which node in OPEN_D to expand next (i.e., what priority function to use) is at the heart of any search algorithm. Since each global bound B_i is essentially a termination condition, expanding nodes that will cause B_i to increase as early as possible would likely result in earlier termination. For example, A* [9] terminates when a solution is found whose cost equals the minimal f-value in OPEN_F ($fMin_F$). Consequently, A* expands a node n with $f(n) = fMin_F$, as this expansion policy is *targeted* toward increasing the f-bound. Similarly, BAE* and DIBBS used the b-bound (B_4) as a termination condition and thus expand nodes n with minimal $b_D(n)$ value. In a similar manner, expansion policies can be defined to target each of the above bounds. For example, the expansion policy that focuses on increasing *KKAdd* (bound B_1) expands a node n with minimal $f_F(n)$-value or the node m with minimal $d_B(m)$, depending on the chosen direction D. In general, each of our bounds has a term to minimize from the forward side and a term to minimize from the backward side, and we choose to expand a node whose bound equals the relevant Min term according to the chosen direction D.

When several global bounds are used, we need to choose which bound to target. Many policies are possible; we describe two of them that we used in our experiments:

Targeting the max This policy targets the bound which has the maximal value among all bounds, B_{max}. The motivation here is that if we manage to raise this bound by expanding nodes, then we have a higher chance that the incumbent solution U will satisfy $cost(U) = B_{max}$, allowing the algorithm to terminate. Thus, we choose to expand a node n in a given direction D whose individual bound $B_{max}(n) = B_{max}$.

Targeting the bound with the smallest span Given a global bound B, we define $span(B)$ to be the number of nodes n with an individual-node bound equal to the value of B. Let B_{ss} be the global bound with the *smallest span* in the set of bounds. This policy chooses to expand a node n whose individual bound is equal to the value of B_{ss}. The intuition for this policy is to focus on the bound that would require the smallest effort to increase.[3]

[3] This is a greedy computation based on the current state of the open lists, as new nodes with the same B_{ss}-values can be generated.

4.3. Choosing which side to expand

Finally, we deal with the decision of whether to expand a node from OPEN_F or OPEN_B at each step. To this end, we explore three policies: *alternating*, *cardinality criterion* [16], and a new policy called *Fastest Bound Increase (FBI)*. The alternating policy switches between forward and backward sides, while the cardinality criterion selects the side with the smallest open list. FBI aims to achieve the fastest increase in a specific bound B_i by comparing the spans of B_i in OPEN_F and OPEN_B and choosing the direction with the smaller span. For policies targeting the maximal bound or minimal span, FBI calculates B_{max} or B_{ss} for both directions and selects the side with the smallest span.

4.4. Targeted bound algorithms

Combining the high-level structure of Algorithm 1 with the 17 newly introduced bounds, we define a class of algorithms that leverage these bounds for both prioritization and termination. We refer to these as *targeted bound* (TB) algorithms, denoted TB_i for $1 \leq i \leq 17$. In TB_i, all 17 bounds are used for termination (Line 3 of Algorithm 1), while only the targeted bound B_i guides node expansion (Line 5). Since TB targets a single bound for prioritization, it does not support smallest-span or max policies, which require simultaneous consideration of multiple bounds for expansion. All TB algorithms have been implemented with the alternating, cardinality, and FBI side-choosing policies. TB_4 provides a direct comparison point with BAE*, as both use the same priority function (minimal b-value); however, BAE* terminates solely with B_4, whereas TB_4 uses all 17 bounds.

4.5. DBBS variants

To further explore the behavior of DBBS in the context of the newly discovered bounds, we introduce two new versions that modify its configuration along two key dimensions: the set of bounds used in the fixed-point computation and the node-selection policy. The first variant, DBBS^{all}, expands upon the original DBBS by incorporating all 17 available lower bounds, rather than the default subset (B_{1-4} and B_{8-9}). This modification allows us to evaluate whether the inclusion of the new bounds has a measurable impact on performance.

In addition to changing the termination bound set, we also investigated alternative node-selection policies. While the original DBBS selects nodes using minimal b-value (and thus denoted DBBS_b), we now consider policies smallest-span (DBBS_ss) and max (DBBS_max). Similar to the TB variants, all DBBS variants have been implemented with the alternating, cardinality, and FBI side-choosing policies

4.6. Lower bound via linear combination

So far, we have focused on the various bounds discussed in Section 3.1, which utilize different subsets of terms from the max-expression in Equation (2) (lb_C). In this section, we will investigate bounds based on linear combinations of these terms, as detailed in Section 3.2. It is worth noting that Equation (3) (lb_{CU}) could also be used in place of Equation (2).

Given the infinite number of possible linear combinations of terms, we need an effective method to select an appropriate combination. Our approach is to first evaluate the individual contribution of each term in the max expression and use this information to guide the combination process. To achieve this, we introduce a variant of the TB_i algorithms, termed *Single-termination Targeted Bound* (STB_i). Similar to TB_i, STB_i selects nodes based on a chosen bound, but unlike TB_i, it terminates using only that specific bound rather than all bounds. This approach allows us to assess each bound's individual contribution more accurately, helping us determine how best to combine them. We focus specifically on the bounds corresponding to the individual terms of the max expression—B_1 (fd-bound), B_2 (df-bound), and B_3 (g-bound)—which yields STB_1, STB_2, and STB_3. First, we execute STB_1, STB_2, and STB_3 on a set of training problems, logging the number of node expansions e_i for each algorithm STB_i. Next, we assign weights according to the performance of each individual bound. We consider two methods for defining these weights: the first method (Equation (8)) normalizes the inverse of the number of expansions, while the second method (Equation (9)) calculates the difference between the total expansions and the number of expansions for each algorithm, normalizing the resulting values.

$$w_i = \frac{e_i^{-1}}{\sum_j e_j^{-1}} \tag{8}$$

$$w_i = \frac{(\sum_j e_j) - e_i}{\sum_k ((\sum_j e_j) - e_k)} \tag{9}$$

Using the calculated weights, we create versions of STB, that target these weights, and run them on the test set. We denote these versions as STB_inv and STB_dif, respectively.

5. Computing the MVC of GMX

To enable a fair comparison of algorithm performance in terms of the minimal number of necessary expansions, it is essential to compute the Minimum Vertex Cover (MVC) of the various GMX graphs. However, the MVC for GMX_C and GMX_CU has not been previously computed in the context of BiHS, and doing so is a non-trivial task. In this section, we present our method for computing the MVC of the different GMX graph variants.

$$\text{Notation:} \quad (g_D, h_D, h_{\overline{D}})$$
$$CLOSED_F = (0,1,0), (1,2,1), (2,1,1), (1,2,1), (3,1,1), (4,0,1)$$
$$CLOSED_B = (0,1,0), (1,2,1), (3,0,0), (2,1,1), (3,1,2), (4,0,1)$$

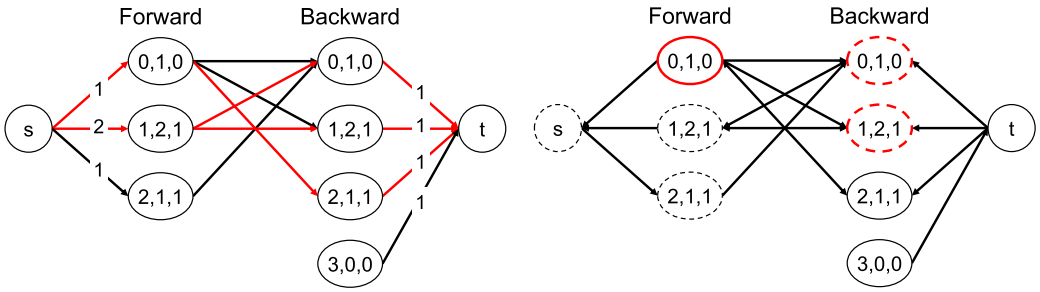


Fig. 1. Example of MVC calculation. The closed lists at the end of the search are shown on top. The flow network constructed based on these lists is shown on the bottom left, in which edges without associated capacity have a capacity of ∞, and edges involved in the max-flow are red. The residual network is shown in the bottom right, where the set of reachable vertices from s is depicted in dashed lines and the MVC is highlighted in red.

To construct the GMX, we first run A* twice: once from *start* to *goal* and once from *goal* to *start*, yielding the optimal solution cost C^* and the closed lists from both searches. We use nodes from these closed lists to form the vertices of the GMX graphs. All GMX variants are bipartite, with one side consisting of nodes from the forward search that have an f-value less than C^*, and the other side symmetrically containing nodes from the backward search. For efficiency, we group nodes on each side into buckets based on identical g_D−, h_D-, and $h_{\overline{D}}$-values, and record the number of nodes in each bucket. An edge is added between two opposite buckets if their corresponding LB value is less than C^*, determined by the specific GMX variant: Equation (1) for the original GMX, Equation (2) for GMX_C, and Equation (3) for GMX_CU.

The MVC of the original GMX (constructed using the lower bound in Equation (1), without assuming heuristic consistency) can be computed efficiently in time linear in the number of buckets [19]. This efficiency arises because the MVC in this case is *strongly restrained*— there exists a fraction p^* such that the MVC is composed of all states with $g_F(n) < p^* \cdot C^*$ from the forward direction and $g_B(n) < (1-p^*) \cdot C^*$ from the backward direction. As a result, the MVC can be found via a linear sweep over all possible partitions p, selecting the one that yields the smallest vertex cover.

In contrast, the graphs GMX_C and GMX_CU do not exhibit the *strongly restrained* property, precluding a similarly efficient computation [20]. To compute the MVC for them, we reduce the problem to a max-flow problem using König's theorem [12]. Specifically, we augment the bipartite graph by introducing a source node s and a sink node t. The source s is connected to all forward-direction buckets, with edge capacities set to the number of nodes in each bucket; similarly, all backward-direction buckets are connected to t using the same capacity scheme. Buckets are connected to one another according to the structure of the corresponding GMX. We then compute the maximum flow from s to t using the algorithm of Boykov and Kolmogorov [3]. The value of the resulting flow corresponds to the size of the MVC. The MVC itself can be extracted from the residual network, which is constructed by including all non-saturated edges of the original flow network and adding reverse edges for those carrying flow. To compute the MVC, let S denote the set of vertices reachable from the source node s in this residual network. The MVC is then given by and returning $U \setminus S \cup (V \cap S)$, where U are the forward buckets and V are the backward buckets. The algorithm has a worst-case time complexity of $O(EV^2|C|)$, where E is the number of edges, V is the number of buckets, and $|C|$ is the size of the minimum cut. Despite this, the computation remains practical for many domains due to the typically small values of $|C|$ and V.

An example of the calculation of the MVC of GMX_CU example can be seen in Fig. 1. In the top of the figure, we see the nodes expanded during the search in each direction, where each node is represented as a tuple of its $(g_D, h_D, h_{\overline{D}})$ values. The bottom left of the figure shows the network flow constructed following the procedure described above. Notably, forward and backward buckets are paired according to their $(g_D, h_D, h_{\overline{D}})$ values, and are connected iff their lb_{CU} value (Equation (3)) is less than $C^* = 4$ (assuming $\epsilon = 0$). The maximum flow, shown in red, has value 3, which implies that the minimum vertex cover (MVC) is also of size 3. To extract the MVC, we first compute the set $S = \{s, (2,1,1)_F, (0,1,0)_B, (1,2,1)_F, (1,2,1)_B\}$ which contains the buckets reachable from s in the residual network shown in the bottom right of the figure (where F denotes a forward bucket and B a backward bucket). The MVC is then $U \setminus S \cup (V \cap S) = \{(0,1,0)_F, (0,1,0)_B, (1,2,1)_B\}$, as shown in red in the figure, confirming a cover of size 3.

6. Empirical evaluation

The empirical evaluation has several objectives. After defining the domains and settings of the evaluation (Section 6.1), we study the theoretical bound on node expansions by comparing the MVCs of the different GMXs. Following this, we evaluate algorithm performance, examining various aspects such as the impact of bounds on search efficiency, node-selection policies, side-choosing criteria, and runtime.

Table 1
Summarization of domains, heuristics used, and number of instances.

Domain	Heuristics	Instances
14-Pancake Problem	GAP and GAP-n	50 random instances
15-Sliding Puzzle	Manhattan Distance	100 standard instances
Grid-based Pathfinding	Octile Distance	3149 DAO instances
12 Disk ToH	Additive PDB	50 random instances
Road Networks	Euclidean distance / max speed	100 random instances

This empirical evaluation serves multiple purposes: understanding the theoretical benefits of utilizing consistency; comparing UniHS and BiHS algorithms; contrasting BiHS variants that assume admissibility with those that assume consistency; assessing the performance of the newly introduced bounds and DBBS variants; analyzing the impact of different side-selection policies; defining and evaluating a lower bound as a linear combination; and comparing the number of nodes expanded by each algorithm to the theoretical minimum.

Notably, all code required to rerun the experiments, including algorithms, domain implementations, experiment scripts, and the problem instances, is publicly available [24].

6.1. Experimental settings

We experimented on five domains, which are listed below and summarized in Table 1: **14-Pancake Puzzle** using the GAP heuristic [10] on 50 random instances. To get a range of heuristic strengths, we also used the GAP-n heuristics (for $1 \leq n \leq 6$) where GAP-n in the forward direction ignores the top n pancakes of $goal$, and in the backward direction it ignores the top n pancakes of $start$. Note that GAP will always return values greater than or equal to GAP-1, which will always return values greater than or equal to GAP-2, and so on. **15 Puzzle (STP)** with the standard 100 instances [13] using the Manhattan distance (MD) heuristic. **Grid-based Pathfinding (Grid)** using octile distance as a heuristic and a cost of 1.5 for diagonal edges: 156 maps from Dragon Age Origins (DAO) [26], each with different start and goal points (a total of 3149 instances). **12-disk Towers of Hanoi (ToH)** with 50 instances with (10+2), (8+4) and (6+6) additive PDBs [8]. And **Road Networks (Road)** with 100 random instances on the map of Colorado [6]. Edges represent driving time between nodes, and the heuristic is the Euclidean distance divided by the maximum speed.

For each algorithm, we report the average number of node expansions required to **terminate** (denoted as $\leq C^*$). In addition, we report the average number of necessary expansions required to **prove optimality** (denoted as $< C^*$), i.e., the number of nodes expanded until the lower bound has reached C^*. Each table reports the arithmetic mean over the applicable instances unless specified otherwise. Each algorithm and specific instance combination was given a runtime limit of 24 hours and a memory limit of 128 GB. This is applicable for the MVC calculations as well. Our experiment was conducted on a computing cluster equipped with AMD EPYC 7763 64-Core processors per machine. While the memory limit is very high, we note that multiple domains can be fully solved by all algorithms with a much lower memory requirement (as low as 8 GB), e.g., ToH and Grid. If an algorithm fails to solve certain instances due to exceeding these limits, the results are presented as #X, where X represents the number of instances successfully solved. If an algorithm fails to solve all instances, or otherwise inapplicable, the outcome is reported as N/A. In cases where the MVC was not computed due to either reaching the memory or time limits in at least one instance, the value is marked N/A. All failures in the STP and Pancake domains were due to memory limits, whereas those in the Grid and Road domains were caused by exceeding the time limit. Importantly, the top-performing algorithms operated well within these constraints—consistently using at least 15 times less memory and runtime than the given limits across all domains—indicating that their superior performance is not a consequence of narrowly bounded computational resources.

6.2. Comparison to the MVC of GMXs

Previous research on BiHS with consistent heuristics has either focused on theoretically studying which nodes must be expanded [20], defining GMX_{CU} without actually computing it, or empirically evaluating algorithms [14,11,17,2,1] without demonstrating their relationship to the actual lower bound on node expansions. Our first experiment aims to bridge this gap by constructing different GMX variants across various problem domains. The experiment serves two purposes: First, it enables us to estimate the reduction in node expansion bounds resulting from each assumption (consistency and undirected graphs). Second, it allows us to assess algorithm performance not only relative to each other but also in comparison to the theoretical bounds, providing insight into how closely the algorithms approach the best performance limit. Specifically, the difference between the MVC and the actual number of nodes expanded by the evaluated algorithms indicates whether new BiHS algorithms that assume consistency should be developed (if there is a significant gap between them) or the focus should be turned elsewhere.

The runtime of the MVC of GMX calculation is almost entirely spent on the A* searches, with the MVC computation taking only a small amount of time (typically under a second), even in domains with many buckets. In contrast, for GMX_C and GMX_{CU}, the MVC computation can dominate the runtime. With a relatively small number of buckets, the A* searches still account for most of the time, but when many buckets are involved, the max-flow algorithm can become the bottleneck.

Table 2 presents the minimum vertex cover (MVC) of GMX, GMX_C, and GMX_{CU} (where applicable within our computation limits) across all domains. In ToH, STP, and Pancake, the MVC of GMX is significantly larger than that of GMX_C—by a factor of 3 for ToH, 7 for STP, and 8–15 for Pancake, except when using the highly accurate GAP heuristic. This demonstrates that the consistency assumption

Table 2
MVC of the GMX for all experimental domains.

	Pancake					
	GAP	GAP-1	GAP-2	GAP-3	GAP-4	GAP-5
MVC(GMX)	39	4,547	82K	452K	N/A	N/A
MVC(GMX_C)	37	297	6,711	59K	N/A	N/A
MVC(GMX_{CU})	37	297	6,711	59K	N/A	N/A
	ToH			STP	Grid	Road
	(10+2)	(8+4)	(6+6)	MD	Octile	$Dist/Speed_{max}$
MVC(GMX)	123K	557K	624K	9,267K	4,290	78K
MVC(GMX_C)	41K	174K	363K	1,308K	N/A	N/A
MVC(GMX_{CU})	41K	174K	363K	1,308K	N/A	N/A

Table 3
Expansions for ToH.

Algorithm	(10+2)		(8+4)		(6+6)	
	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$
A*	276K	276K	1,926K	1,925K	3,268K	3,239K
rA*	123K	123K	622K	620K	1,064K	1,033K
NBS	230K	230K	647K	645K	682K	662K
DVCBS	232K	232K	619K	609K	663K	635K
BAE*(a)	**47K**	**46K**	**187K**	**186K**	**383K**	382K
$DBBS_b$(a)	48K	**46K**	189K	**186K**	**383K**	**379K**
TB_1(a)	54K	53K	204K	204K	424K	423K
TB_2(a)	69K	69K	240K	240K	435K	435K
TB_3(a)	648K	622K	662K	660K	683K	664K
TB_4(a)	**47K**	**46K**	**187K**	**186K**	**383K**	382K
TB_5(a)	263K	262K	386K	382K	515K	509K
TB_6(a)	282K	277K	412K	406K	513K	512K
TB_7(a)	170K	165K	315K	308K	458K	455K
TB_{14}(a)	1,749K	1,671K	1,484K	1,480K	1,211K	1,208K
$DBBS_{ss}$(a)	51K	49K	197K	196K	399K	398K
$DBBS_{max}$(a)	48K	**46K**	192K	189K	390K	387K
$DBBS^{all}_b$(a)	48K	**46K**	189K	**186K**	**383K**	**379K**
$DBBS^{all}_{ss}$(a)	52K	50K	198K	197K	403K	402K
$DBBS^{all}_{max}$(a)	48K	**46K**	192K	189K	390K	387K

greatly reduces the number of required expansions in these domains. For Road Networks and Grid, calculating the MVC of GMX_C was prohibitively expensive due to high solution costs and many unique buckets. Nonetheless, the small performance difference between algorithms assuming only admissibility and those also assuming consistency, as will be demonstrated in the following section, suggests that in these domains the MVCs of GMX, GMX_C, and GMX_{CU} are similar, with perhaps a slight advantage for consistency on road networks.

Additionally, the MVCs of GMX_C and GMX_{CU} are of equal size, indicating that the new condition introduced in Equation (3) might not lead to better performance compared to bounds derived from Equation (2).

In the following sections, we evaluate the algorithms' performance and compare their results to the MVC of the corresponding GMX.

6.3. Comparison of algorithm performance

In this section, we evaluate and compare the performance of various TB algorithms, each targeting a specific bound, along with DBBS variants with the different node-selection policies (as detailed in Section 4.2).

For our baselines, we selected **A*** as a representative UniHS algorithm, along with a reverse version of A* that operates from *goal* to *start*, referred to as **rA***. Additionally, we included **NBS** and **DVCBS** as representative BiHS algorithms that assume heuristic admissibility but not consistency. Lastly, we used **BAE***, the current state-of-the-art BiHS algorithm, alongside the original DBBS algorithm, both of which rely on heuristic consistency. Since the original DBBS algorithm always expands nodes with minimal b-value, we denote it as $DBBS_b$ to differentiate it from the other variants.

For brevity, we begin by focusing on the alternating side-choosing policy for selected algorithms (denoted (a)). Nonetheless, Section 6.4 analyzes different side-choosing criteria, and the full results, including all criteria, are available in Appendix C. Additionally, we begin by using node expansions as the primary performance measure, but later in Section 6.4.3, we expand the analysis to include runtime evaluation.

Table 4
Expansions for Pancake.

Algorithm	**GAP**		**GAP-1**		**GAP-3**		**GAP-5**	
	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$
A*	**72**	**46**	8,423	8,400	6,838K	6,834K	#25	
rA*	77	52	7,077	6,854	6,906K	6,830K	#26	
NBS	148	66	6,477	6,220	699K	699K	1,623K	1,623K
DVCBS	209	47	4,885	4,635	492K	479K	1,137K	1,039K
BAE*(a)	90	66	654	552	132K	113K	1,116K	1,087K
$DBBS_b$(a)	114	57	688	458	77K	70K	653K	640K
TB_1(a)	136	86	1,406	1,330	2,042K	1,879K	10,232K	10,133K
TB_2(a)	149	100	1,640	1,446	1,758K	1,755K	11,337K	11,020K
TB_3(a)	484K	120K	657K	494K	1,623K	1,623K	1,624K	1,623K
TB_4(a)	90	66	654	552	132K	113K	1,116K	1,087K
TB_5(a)	27K	16K	60K	48K	353K	336K	1,006K	972K
TB_6(a)	26K	16K	52K	46K	369K	341K	1,137K	1,010K
TB_7(a)	2,724	2,073	6,636	5,861	114K	106K	596K	555K
TB_{14}(a)	#16		#21		#39		#42	
$DBBS_{ss}$(a)	113	57	634	457	80K	78K	526K	502K
$DBBS_{max}$(a)	114	57	688	458	77K	70K	653K	640K
$DBBS^{all}_b$(a)	114	57	688	458	77K	**69K**	639K	626K
$DBBS^{all}_{ss}$(a)	113	57	**628**	**450**	**75K**	73K	**515K**	**485K**
$DBBS^{all}_{max}$(a)	114	57	688	458	77K	**69K**	523K	508K

Table 5
Expansions for STP, Grid, and Road.

Algorithm	**STP**		**Grid**		**Road**	
	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$
A*	15,550K	14,700K	5,406	5,322	127K	127K
rA*	11,144K	10,956K	**5,342**	5,267	126K	126K
NBS	12,556K	11,739K	6,873	6,556	96K	96K
DVCBS	10,930K	10,720K	5,545	**5,151**	126K	126K
BAE*(a)	2,707K	2,700K	6,718	6,668	**67K**	**67K**
$DBBS_b$(a)	**2,262K**	**1,701K**	6,105	5,829	N/A	
TB_1(a)	13,404K	12,953K	7,081	6,940	70K	70K
TB_2(a)	10,815K	10,677K	7,150	7,018	72K	72K
TB_3(a)	#60		10,275	8,654	119K	119K
TB_4(a)	2,707K	2,700K	6,706	6,483	**67K**	**67K**
TB_5(a)	#95		9,213	7,955	96K	96K
TB_6(a)	#96		9,221	8,021	96K	96K
TB_7(a)	13,448K	13,275K	8,562	7,563	86K	86K
$DBBS_{ss}$(a)	2,306K	1,954K	6,092	5,794	N/A	
$DBBS_{max}$(a)	2,646K	2,041K	6,261	5,972	N/A	
$DBBS^{all}_b$(a)	**2,262K**	**1,701K**	6,104	5,827	N/A	
$DBBS^{all}_{ss}$(a)	2,292K	1,935K	6,094	5,797	N/A	
$DBBS^{all}_{max}$(a)	2,646K	2,041K	6,163	5,829	N/A	

The results are organized into tables by domain. Table 3 presents the results for Towers of Hanoi (ToH), Table 4 shows the results for the Pancake domain, and Table 5 covers the Sliding Tile Puzzle (STP), grid maps, and road networks. Below, we analyze the different trends presented in the results across the domains.

6.3.1. The effect of assuming heuristic consistency

When comparing baseline BiHS algorithms that assume heuristic consistency, namely BAE* and $DBBS_b$, with those that do not, NBS and DVCBS, we observe a clear advantage in utilizing consistency. For instance, in the Towers of Hanoi (ToH), the algorithms that do not assume consistency require between 1.8X and 5X more node expansions to solve the problem than those that do. In the Sliding Tile Puzzle (STP), the advantage is also 5X, while in the Pancake domain, this advantage can reach up to 10X when using the GAP-1 heuristic. However, in polynomial domains like grid maps and road networks, the advantage is less pronounced, with no significant benefit observed in grid environments. These results align with previous findings [2] and are consistent with the analysis of the MVC of the different GMX variants presented in Section 6.2.

6.3.2. UniHS vs BiHS

First, we observe that in some domains, such as Pancake, Grid, and Road, A* and rA* exhibit similar performance. However, in other domains, like STP and ToH, rA* outperforms A* significantly. This difference arises from the asymmetric branching factor in both search directions. For example, in STP, the blank tile starts in the corner at the goal state, allowing only two possible moves from $goal$ instead of the usual four.

When excluding BAE* and $DBBS_b$, which assume consistency, the performance of the UniHS and BiHS algorithms is largely comparable, with each group exhibiting a slight advantage in different domains. Notably, when the heuristic is weak—such as the (6+6) Pattern Database (PDB) heuristic in the Towers of Hanoi (ToH) and the GAP-X heuristics in the Pancake domain—the BiHS algorithms demonstrate a clear advantage. In particular, with the GAP-5 heuristic, both A* and rA* manage to solve only a limited number of problems within the memory constraints. Overall, among the two BiHS algorithms, NBS and DVCBS, the latter consistently performs better, aligning with the findings of Shperberg et al. [23].

When including BAE* and $DBBS_b$, BiHS algorithms demonstrate a clear advantage in nearly every domain and heuristic, except for the Pancake problem with the GAP heuristic and grid maps.

6.3.3. Targeted bound algorithms: end-to-end analysis

In this section, we compare the various targeted bound algorithms and analyze the impact of the different bounds. We begin by analyzing the end-to-end performance of the various TB algorithms, followed by an examination of how the different bounds evolve throughout the search.

Our tables only present TB variants for the first seven bounds; however, the results for all 17 bounds are available in the appendix. Overall, the remaining ten bounds TB_8 to TB_{17}) did not demonstrate effectiveness. In some cases, such as TB_8, TB_9, TB_{12}, TB_{13}, TB_{14}, and TB_{17} in the Towers of Hanoi (ToH) and Pancake domain, the number of nodes expanded by these algorithms tends to decrease as heuristic accuracy declines.

It is important to note that rf (reverse f) behaves almost like the inverse of f. Let us consider two nodes with a g-value of 5. One is closer to $goal$ and has a heuristic of 3, while the other, being further away, has a heuristic of 5. Expanding based on minimal rf would select the node further from the goal, as its priority is 0, compared to a priority of 2 for the closer node. This implies that with a more accurate heuristic, an algorithm using minimal rf would prefer nodes that are *further* from $goal$ rather than closer. This would suggest that as the heuristic degrades, the algorithm would be closer to a search purely based on g-values and thus would improve. Similarly, rd prefers nodes where the reverse heuristic is low, i.e., has high heuristic error. This suggests that employing reverse values, specifically utilizing rf and rd, generally undermines the search process, indicating that relying on them alone is inadvisable. These results are consistent with those from the previous section, where we observed that the MVC of GMX_C and GMX_{CU} are of the same size, indicating that the additional bounds are unlikely to offer any significant advantage.

Overall, when considering ToH, road networks, and STP, TB_4 has the best performance among all other Targeted Bound Algorithms, and by extension also BAE*. In Grid, TB_1, TB_2, and TB_4 can be considered the best algorithms. In the Pancake domain, when employing a highly accurate heuristic such as GAP, unidirectional search demonstrates the best performance. However, with GAP-1, TB_4 emerges as the top-performing algorithm. As the heuristic quality declines (starting from GAP-3), TB_7 takes the lead, expanding significantly fewer nodes than TB_4—almost half the number of nodes expanded by TB_4 with GAP-5. This observation suggests that, in certain domains, prioritizing the g-value can yield superior results when the heuristic is weak.

6.3.4. Targeted bound algorithms: bound analysis throughout the search

Recall that each TB_i algorithm uses the target bound B_i to guide node expansion, but relies on all available bounds for termination—not just the targeted one. In this section, we examine whether incorporating multiple bounds for termination leads to earlier termination, thereby reducing the number of node expansions. We discuss the associated overhead separately in Section 6.4.3.

Fig. 2 illustrates the average values of each of the seven bounds (as a fraction of C^*) during the search in the Pancake domain, employing two targeted bounds: TB_4 (the b-bound, left) and TB_7 (right), alongside two heuristics, GAP-1 (top) and GAP-5 (bottom). In each plot, the uppermost bound represents the dominating bound, and the search terminates once this bound equals C^*. GAP-1 is used instead of GAP because GAP is such a strong heuristic that it is difficult to draw meaningful trends and bound progress because the searches themselves are too short.

Figs. 2c and 2d demonstrate that for GAP-5, the targeted bound (bound B_4 for TB_4 and B_7 for TB_7) consistently maintains the highest value throughout the search. This indicates that the targeted-bound policy successfully fulfills its objective of rapidly increasing the bound of interest. However, Fig. 2b reveals that when a non-targeted bound, such as B_4, significantly outperforms the targeted bound, it might hold the highest value during the search, even over the targeted bound. Fig. 2a reinforces the notion that B_4 is a strong bound, as when targeting it with a strong heuristic, unlike B_7, the other bounds are not as close to it.

Table 6 shows the percentage of solved instances where the search was terminated by the targeted bound.

In ToH, even with a highly accurate heuristic, nearly all TBs targeting bounds 1–7 (except for TB_3) were terminated by their respective bound. In contrast, TBs targeting bounds 8–17 were often terminated by other bounds. This reinforces the earlier claim that, with a strong heuristic—especially under inefficient prioritization—alternative bounds can serve as effective termination criteria. As the heuristic weakens, the targeted bound becomes dominant, terminating all solved instances (except in the N/A case, where no instances were solved). This pattern also holds in STP.

In the pancake domain, this trend is most pronounced. When using GAP, most TB algorithms rarely terminate with their targeted bound, particularly for bounds 8–17, which almost never terminate via their respective bound. Once again, as heuristic quality decreases, more TBs terminate with their respective bounds.

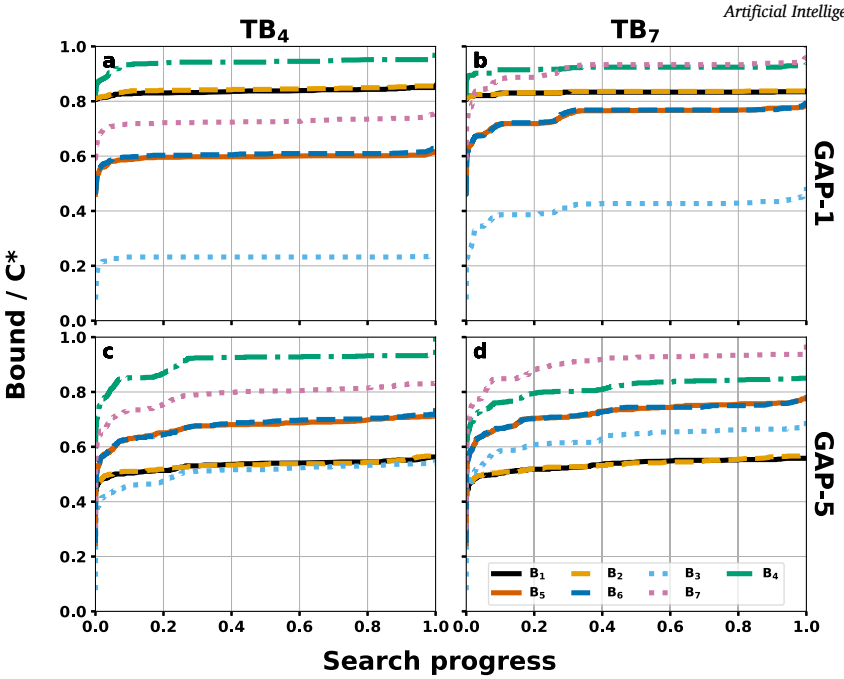


Fig. 2. Value of different bounds as a function of search progress.

Table 6
% of solved instances where the terminating bound was the targeted bound.

Alg	ToH			STP	Pancake					
	10+2	8+4	6+6	MD	GAP	GAP-1	GAP-2	GAP-3	GAP-4	GAP-5
TB_1(a)	100%	100%	100%	100%	100%	84%	100%	100%	100%	100%
TB_2(a)	100%	100%	100%	100%	100%	80%	100%	100%	100%	100%
TB_3(a)	62%	94%	100%	100%	48%	52%	90%	100%	100%	100%
TB_4(a)	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
TB_5(a)	100%	100%	100%	100%	84%	96%	100%	100%	100%	100%
TB_6(a)	100%	100%	100%	100%	74%	92%	98%	100%	100%	100%
TB_7(a)	100%	100%	100%	100%	100%	98%	100%	100%	100%	100%
TB_8(a)	42%	96%	100%	N/A	0%	0%	33%	73%	93%	100%
TB_9(a)	66%	94%	100%	N/A	0%	0%	30%	76%	96%	100%
TB_{10}(a)	70%	100%	100%	100%	0%	0%	34%	88%	100%	100%
TB_{11}(a)	66%	100%	100%	100%	0%	0%	40%	90%	98%	100%
TB_{12}(a)	56%	100%	100%	100%	0%	6%	67%	88%	100%	100%
TB_{13}(a)	76%	96%	100%	100%	0%	6%	67%	90%	100%	100%
TB_{14}(a)	28%	86%	100%	N/A	0%	0%	26%	59%	88%	98%
TB_{15}(a)	74%	96%	100%	100%	4%	16%	74%	98%	100%	100%
TB_{16}(a)	60%	96%	100%	100%	6%	12%	66%	98%	100%	100%
TB_{17}(a)	34%	92%	100%	100%	0%	0%	22%	48%	78%	88%

Nevertheless, while the targeted bound often dominates toward the end of the search—especially with weaker heuristics—this is not always the case. When using stronger heuristics and inefficient prioritization, termination by non-targeted bounds is common, particularly for TBs targeting bounds 8-17. Thus, tracking other bounds can be useful in such cases, despite the incurred overhead.

These results are consistent with the results of BAE* and TB_4 in the previous section, showing that both algorithms perform similarly in most domains, as TB_4 does not benefit from any of the other bounds for termination except the *b*-bound. The only exception is in the grid domain, in which TB_4 expands fewer nodes than BAE*, suggesting that in this domain, keeping track of other bounds can be helpful, though by an extremely small margin.

The data further highlights the performance gap between bounds 1–7 and bounds 8–17. For bounds 1–7, the targeted bound is almost always responsible for terminating the search. In contrast, this is not the case for bounds 8–17, suggesting that these bounds are inherently weaker.

6.3.5. DBBS variants

First, when comparing the standard DBBS variants to the $DBBS^{all}$ variants across all node-selection policies, we observe that their performance is nearly identical, with only minimal differences in node expansions across domains. This aligns with our earlier

Table 7
Expansions for Pancake.

Algorithm	GAP		GAP-1		GAP-3		GAP-5	
	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$
BAE*(a)	90	66	654	552	132K	113K	1,116K	1,087K
BAE*(p)	88	64	730	621	154K	135K	1,116K	1,087K
BAE*(FBI)	**75**	**39**	**427**	**317**	92K	77K	869K	834K
TB_1(a)	136	86	1,406	1,330	2,042K	1,879K	10,232K	10,133K
TB_1(p)	150	100	6,766	6,712	3,691K	3,028K	11,371K	11,063K
TB_1(FBI)	#21		#9		#49		8,047K	7,521K
TB_2(a)	149	100	1,640	1,446	1,758K	1,755K	11,337K	11,020K
TB_2(p)	154	101	6,361	6,325	3,099K	2,780K	13,428K	12,363K
TB_2(FBI)	#24		#8		#47		8,864K	7,892K
TB_4(a)	90	66	654	552	132K	113K	1,116K	1,087K
TB_4(p)	88	64	730	621	154K	135K	1,116K	1,087K
TB_4(FBI)	**75**	**39**	**427**	**317**	92K	77K	869K	834K
TB_7(a)	2,724	2,073	6,636	5,861	114K	106K	596K	555K
TB_7(p)	2,862	2,209	6,173	5,397	117K	109K	595K	554K
TB_7(FBI)	3,704	1,633	6,705	4,793	101K	86K	498K	456K
$DBBS_{ss}$(a)	113	57	634	457	80K	78K	526K	502K
$DBBS_{ss}$(p)	146	43	542	329	73K	68K	**481K**	454K
$DBBS_{ss}$(FBI)	104	43	517	353	**69K**	67K	489K	**452K**
$DBBS_b^{all}$(a)	114	57	688	458	77K	69K	639K	626K
$DBBS_b^{all}$(p)	152	42	610	326	72K	64K	579K	559K
$DBBS_b^{all}$(FBI)	103	42	601	326	71K	**63K**	603K	564K

analysis, suggesting that the additional 10 bounds offer little to no contribution. We now turn to comparing the various node-selection policies.

The difference in node expansions across all node-selection policies for DDBS is generally small—typically within 10% in most domains, with a maximum difference of 28%. In the Grid and Pancake domains, $DBBS_{ss}$ achieved the best performance, whereas in STP and ToH, $DBBS_b$ performed the best. Interestingly, the max node-selection policy never led to the best performance. The initial hypothesis was that this policy would allow algorithms to dynamically switch to a more promising bound, adapting to each instance. However, our analysis shows that once the algorithm begins focusing on a specific bound, it tends to continue expanding nodes based on that bound. This observation aligns with the results in Section 6.3.3. Consequently, at the early stages of the search, when all bounds are similar, the max-policy can converge on a less effective bound and continue to focus on it throughout the search.

6.4. Side-choosing criteria

In this section, we will look at the effect of each side-choosing policy: alternating (a), Pohl's Cardinality (p), and the new fastest bound increase (FBI). To enhance clarity, the tables in this section focus solely on the TB variants that demonstrated the best performance in at least one domain. For the same reason, we focus on DBBS with only the two node prioritization schemes that produced competitive results, smallest-span and minimal b-value. While the trends are consistent across all algorithms, comprehensive results, including all algorithms, are available in the appendix.

Table 7 presents the results for the Pancake domain. For BAE* and its related variants (TB_4 and TB_7), the difference across all policies is within a factor of 2. Notably, the newly introduced FBI policy performs better than the others in almost all cases.

However, the results for TB_1 and TB_2 tell a different story. The alternating policy significantly outperforms Pohl's cardinality approach, which can result in up to five times more node expansions. For these algorithms, FBI performs even worse, initially failing to solve all problem instances despite having access to a highly accurate heuristic. The limitation of FBI lies in its short-sightedness: it selects which bound to prioritize based solely on the current nodes in OPEN, without considering the future nodes that will be generated with the same priority.

While this myopic behavior works well for bounds like b (B_4) and bg (B_7), which tend to increase consistently from parent to child, TB_1 and TB_2 rely on heuristic errors (d_B and d_F, respectively), which are less likely to increase. Specifically, when applying FBI to TB_1, we observed that there are fewer nodes with minimal d_B than with minimal f_F. As a result, FBI prioritizes expanding the backward size to increase the minimal d-value (which is typically 0). However, expanding a node with a d-value often results in at least one successor with the same d-value, meaning the number of nodes with minimal d-value in the open list remains unchanged. While a similar phenomenon can occur with f and b, the issue is less pronounced, as the g-value always increases between successors (due to the minimal edge cost being 1), meaning fewer expansions are needed to expand all nodes with minimal f- or b-values. Consequently, FBI guides the search based on the d_F (or d_B) values, which in these cases are not reliable indicators of future performance. As the heuristic degrades, fewer d-values remain 0, allowing the algorithm to escape this myopic behavior. Thus, FBI performs better on TB_1 and TB_2 when the heuristic is less accurate.

These findings highlight the need for a more informed approach that estimates which direction will require fewer expansions—one that goes beyond immediate priorities and adapts to observed search patterns. Such an approach should dynamically adjust based

Table 8
Expansions for ToH and Grid.

Algorithm	Towers of Hanoi						Grid	
	(10+2)		(8+4)		(6+6))		Octile	
	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$
BAE*(a)	47K	46K	187K	186K	383K	382K	6,718	6,668
BAE*(p)	46K	45K	185K	182K	375K	374K	5,996	5,862
BAE*(FBI)	**45K**	**43K**	**181K**	**178K**	**368K**	**366K**	6,207	6,161
TB_1(a)	54K	53K	204K	204K	424K	423K	7,081	6,940
TB_1(p)	50K	50K	200K	200K	423K	421K	5,823	5,461
TB_1(FBI)	**45K**	45K	190K	190K	404K	402K	6,652	6,481
TB_2(a)	69K	69K	240K	240K	435K	435K	7,150	7,018
TB_2(p)	60K	60K	221K	221K	408K	407K	5,861	5,465
TB_2(FBI)	56K	56K	212K	210K	396K	394K	6,658	6,471
TB_4(a)	47K	46K	187K	186K	383K	382K	6,706	6,483
TB_4(p)	46K	45K	185K	182K	375K	374K	5,986	5,634
TB_4(FBI)	**45K**	**43K**	**181K**	**178K**	**368K**	**366K**	6,197	5,889
TB_7(a)	170K	165K	315K	308K	458K	455K	8,562	7,563
TB_7(p)	165K	158K	309K	303K	451K	446K	7,939	6,806
TB_7(FBI)	168K	159K	310K	300K	448K	440K	8,172	6,909
$DBBS_{ss}$(a)	51K	49K	197K	196K	399K	398K	6,092	5,794
$DBBS_{ss}$(p)	46K	44K	182K	179K	371K	369K	5,660	5,384
$DBBS_{ss}$(FBI)	48K	47K	194K	192K	395K	393K	5,374	**4,935**
$DBBS_b^{all}$(a)	48K	46K	189K	186K	383K	379K	6,104	5,827
$DBBS_b^{all}$(p)	**45K**	**43K**	**181K**	**178K**	371K	**366K**	5,688	5,394
$DBBS_b^{all}$(FBI)	**45K**	**43K**	**181K**	**178K**	370K	**366K**	**5,308**	4,945

Table 9
Linear Combination Expansions in the Pancake Domain.

Algorithm	GAP		GAP-1		GAP-2		GAP-3		GAP-4		GAP-5	
	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$
STB_1(a)	149	89	2,039	1,965	110K	108K	2,792K	2,773K	7,397K	6,767K	11,592K	11,361K
STB_2(a)	148	106	1,780	1,780	123K	123K	3,303K	3,133K	7,198K	6,892K	12,378K	11,743K
STB_3(a)	2,067K	1,912K	2,067K	1,912K	2,071K	1,912K	2,071K	1,912K	2,071K	1,912K	2,071K	1,912K
TB_4(a)	**90**	66	654	552	15K	11K	132K	113K	469K	465K	1,116K	1,087K
TB_7(a)	2,724	2,073	6,636	5,861	35K	33K	**114K**	106K	301K	282K	596K	555K
STB_{inv}(a)	114	**57**	**626**	**404**	**11K**	**8,538**	140K	113K	397K	363K	657K	577K
STB_{dif}(a)	114	**57**	690	452	**11K**	8,636	131K	**102K**	**291K**	**260K**	**499K**	**461K**

on the patterns in node generation, helping the algorithm avoid short-sighted decisions driven by unreliable indicators of remaining nodes.

In Table 8 we can see the results for the same set of algorithms for the ToH domain. For BAE* and the various TB algorithms, the FBI side-choosing policy is shown to be the most effective, exhibiting none of the pathological behavior observed in the Pancake domain. This highlights the importance of matching both the algorithm and the side-choosing policy to the problem domain.

Additionally, Table 8 includes results for the Grid domain, where different side-choosing policies impact algorithm performance. For instance, with the alternating policy, TB_4 expands the fewest nodes, while using Pohl's cardinality favors TB_1. This further emphasizes the significance of selecting the appropriate side-choosing criterion.

6.4.1. Lower bound via linear combination

To evaluate this new method, we conduct experiments in the Pancake and ToH domains, which feature a variety of heuristics that previously yielded different best-performing algorithms. We began by running STB_1, STB_2, and STB_3 on 50 random problems, followed by computing the weights as outlined earlier. Subsequently, we executed STB_{inv} and STB_{dif} on the same 50 test problems utilized in the previous experiments.

The results for Pancake are presented in Table 9, which also includes TB_4 and TB_7, the best-performing targeted bound algorithms in the Pancake domain. The findings indicate that no single weighting scheme consistently outperforms across all heuristics. Specifically, TB_4 excels for GAP, while STB_{inv} is superior for GAP-1 and GAP-2. In GAP-3, TB_7 performs better, and from GAP-4 onwards, STB_{inv} takes the lead. This suggests that even when attempting to tailor specific weightings to a particular set of problem instances, choosing the optimal weighting among the terms remains challenging. Nevertheless, both STB_{inv} and STB_{dif} demonstrate overall strong performance across all domains and heuristics, avoiding significant underperformance in scenarios where the two TB bounds falter—specifically, TB_4 underperforms in GAP-4 and GAP-5, while TB_7 lags in GAP, GAP-1, and GAP-2.

Table 10
Linear Combination Expansions in the ToH Domain.

Algorithm	(10+2)		(8+4)		(6+6)	
	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$
STB_1(a)	56K	56K	207K	207K	431K	430K
STB_2(a)	78K	77K	243K	243K	433K	433K
STB_3(a)	678K	671K	677K	671K	677K	671K
TB_4(a)	**47K**	**46K**	**187K**	**186K**	**383K**	**382K**
TB_7(a)	170K	165K	315K	308K	458K	455K
STB_{inv}(a)	57K	54K	235K	226K	430K	424K
STB_{dif}(a)	69K	65K	259K	248K	441K	433K

Table 11
Linear Combination Weights for Pancake and ToH Domains.

Bounds		ToH			Pancake					
		10+2	8+4	6+6	GAP	GAP-1	GAP-2	GAP-3	GAP-4	GAP-5
STB_{inv}	B_1	0.5547	0.4637	0.3801	0.4972	0.4660	0.5124	0.3131	0.1786	0.1327
	B_2	0.3995	0.3948	0.3781	0.5028	0.5336	0.4603	0.2647	0.1835	0.1243
	B_3	0.0458	0.1415	0.2418	0.0000	0.0005	0.0273	0.4222	0.6379	0.7430
STB_{dif}	B_1	0.4655	0.4083	0.3602	0.5000	0.4995	0.4761	0.3290	0.2781	0.2774
	B_2	0.4521	0.3923	0.3595	0.5000	0.4996	0.4734	0.2978	0.2841	0.2623
	B_3	0.0823	0.1995	0.2803	0.0001	0.0009	0.0506	0.3732	0.4379	0.4602

The results for ToH are presented in Table 10, with TB_4 and TB_7 included as reference points. In this domain, unlike in Pancake, TB_4 consistently emerges as the top-performing algorithm. This suggests that fixed weighting methods may not fully capture the domain-specific dynamics required for effective prioritization. More adaptive approaches—such as reinforcing learning or other hyperparameter tuning mechanisms that adjust weights based on feedback—might offer improvements, though they fall outside the class of methods explored in this work.

Table 11 shows the relative weighting of each bound in the priority functions of STB_{inv} and STB_{dif}, rounded to four decimal places. As expected, weaker heuristics lead to a higher weight on B_3; for example, in ToH, the weight of B_3 increases from 0.046 to 0.24 when moving from the 10+2 PDB to the 6+6 PDB, and in the Pancake problem it rises from 0 to 0.74 when comparing GAP to GAP-5. This increase is mostly beneficial for the Pancake problem, producing a stronger priority function that solves instances with fewer expansions. Generally, STB_{dif} assigns more weight to B_3 than inv. However, in ToH, both priority functions are outperformed by TB_4 for all heuristics, especially STB_{dif}, as it assigns more weight to B_3 than STB_{inv}. This suggests that there are finer details to the optimal weighting of the three bounds than simply looking at their individual expansions, and more careful thought needs to be put into choosing the weighting of the priority function.

6.4.2. Performance of algorithms compared to the bound

To evaluate how the number of nodes expanded by the algorithms compares to the theoretical lower bounds, Table 12 summarizes the performance of the baselines along with selected top-performing algorithms relative to the lower bound MVC(GMX_{CU}). In ToH and road networks, BAE* had the best performance among all algorithms, similar to $DBBS_b^{all}$ (except for road networks, in which the DBBS variants could not complete execution due to the 24 h runtime limit). Moreover, the difference between the number of nodes expanded by BAE* and the MVC of GMX_{CU} is very small ($5-15\%$). Therefore, BAE* is very close to optimal, and no other algorithm (UniHS or BiHS) can significantly improve upon it. Finally, we see that the performance of BAE* is identical to TB_4, which means that the b-bound was always the tightest and never benefited from the other termination criteria.

In STP, BAE* and TB_4 were the best among the bounds that do not make use of fixed-point computation, improving over A*, rA*, NBS and DVCBS by a factor > 4. Here too, the additional stopping criteria of TB_4 did not improve the performance over BAE*. The DBBS variants, which perform the fixed-point computation, improved over BAE* by approximately 25%, where the best-performing variant was $DBBS_b$(FBI) in terms of expansions. In fact, $DBBS_{ss}$(FBI) expands only 28% more nodes than the MVC of GMX_{CU} before raising LB to C^*. Thus, there is only a small margin left for improvement in STP as well.

In contrast to ToH, STP, and road networks, BAE* does not perform well on Grid. In this domain, rA* required the least expansions to return a solution, and $DBBS_{ss}$(FBI) required only a few additional expansions ($\leq C^*$), and fewer necessary expansions ($< C^*$). Notably, the additional termination conditions of TB_4 show a (small) improvement over BAE*, which means that the b-bound is not always dominating in Grid, even when the node-expansion policy only targets b.

6.4.3. Runtime analysis

For this experiment, we used Grid and STP as polynomial and exponential representative domains, respectively. We aim to answer the following questions: (1) What is the overhead of using all bounds for termination compared to using a single bound? (2) What is the overhead of the fixed-point computation? and (3) How does the overhead of the BiHS algorithms compare to A*? Table 13

Table 12
Expansions for ToH, STP, Grid, and Road.

Algorithm	Towers of Hanoi						STP		Grid		Road	
	(10+2)		(8+4)		(6+6))		MD		Octile		Dist/Speed$_{max}$	
	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$
A*	276K	276K	1,926K	1,925K	3,268K	3,239K	15,550K	14,700K	5,406	5,322	127K	127K
rA*	123K	123K	622K	620K	1,064K	1,033K	11,144K	10,956K	5,342	5,267	126K	126K
NBS	230K	230K	647K	645K	682K	662K	12,556K	11,739K	6,873	6,556	96K	96K
DVCBS	232K	232K	619K	609K	663K	635K	10,930K	10,720K	5,545	5,151	126K	126K
BAE*(a)	47K	46K	187K	186K	383K	382K	2,707K	2,700K	6,718	6,668	**67K**	**67K**
MVC(GMX$_{CU}$)		41K		174K		363K		1,308K		N/A		N/A
TB$_4$(a)	47K	46K	187K	186K	383K	382K	2,707K	2,700K	6,706	6,483	**67K**	**67K**
DBBS$_{ss}$(FBI)	48K	47K	194K	192K	395K	393K	2,027K	1,677K	5,374	**4,935**	N/A	
DBBS$_b$(FBI)	**45K**	**43K**	**181K**	**178K**	**370K**	**366K**	**1,977K**	**1,314K**	**5,308**	4,945	N/A	
DBBS$_b^{all}$(FBI)	**45K**	**43K**	**181K**	**178K**	**370K**	**366K**	**1,977K**	**1,314K**	**5,308**	4,945	N/A	

Table 13
Runtime Results of Representative Algorithms.

Algorithm	STP		DAO	
	Time (s)	n/s	Time (ms)	n/ms
A*	55.42	281K	2.28	2371.12
BAE*(a)	10.07	269K	4.08	1646.61
TB$_4$(a)	59.99	45K	87.69	76.48
DBBS$_b$(a)	34.85	65K	2,390.74	2.55

reports the results of this experiment, showing that BAE* is between 6 and 20 times faster than TB$_4$. This speedup is primarily due to the overhead incurred when maintaining the additional bounds (a sorted vector of pointers for each bound). This suggests that TB$_4$ should never be used, as its greater runtime overhead does not justify the small reduction in node expansions. As for the second question, the runtime of DBBS (i.e., the fixed-point computation) in Grid (polynomial domain) is several orders of magnitude greater than other algorithms, while in STP (exponential domain) DBBS is only four times slower in terms of node expansions per second. Nonetheless, even in STP we see that BAE* is faster than DBBS in terms of total runtime, despite expanding more nodes. Finally, we see that A* is only slightly faster than BAE* in terms of expansions per second, thus the total runtime of the two algorithms mostly depends on their respective number of expansions.

As a general guideline, based on the empirical evaluation, BAE* should be the default algorithm to use for the consistency case, as it strikes a good balance between node expansions and runtime across different domains and heuristics. For weaker heuristics (e.g., as seen in GAP-4), we suggest using a targeted-bound algorithm that uses only B_7. In either case, only the targeted-bound should be used for termination, as the utilization of other bounds usually does not lead to less node expansions, or very few, while incurring significant overhead. While in some domains the DBBS variants expand the least number of nodes, the overhead of the fixed-point computation is not justified in terms of total runtime.

7. Summary and conclusions

This paper establishes a direct connection between Must-Expand Pairs (MEP) theory and existing search bounds, showing that all known bounds can be derived from MEP and that infinitely many new bounds can be generated. We introduced the parametric TB algorithm, which takes a specific bound as input and targets it throughout the search. Our results indicate that the b-bound (B_4) is the most informative for most domains. However, the bg-bound (B_7), which incorporates both b and g, proves more effective in cases with weaker heuristics. Additionally, we extended the DBBS algorithm to incorporate new search bounds, alongside novel node selection and search-direction strategies. While these DBBS variants often achieved the lowest node expansions, they also incurred significant computational overhead, leading to increased runtimes.

Furthermore, we proposed a method for learning efficient bounds by solving a subset of instances within a domain and generalizing these insights to new instances using two different approaches. Our comparisons between the expansions required by algorithms to prove a solution's optimality and the theoretical minimum necessary expansions suggest that there is limited room for further optimization. Consequently, research in BiHS can now shift focus away from developing additional algorithms under the consistency assumption and towards other avenues for enhancing search efficiency.

CRediT authorship contribution statement

Lior Siag: Writing – original draft, Software, Methodology. **Shahaf S. Shperberg:** Writing – review & editing, Writing – original draft, Methodology, Formal analysis.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Shahaf Shperberg reports financial support was provided by Israel Science Foundation. Shahaf Shperberg reports financial support was provided by Israel Ministry of Innovation Science & Technology. Shahaf Shperberg reports a relationship with United States-Israel Binational Science Foundation that includes: funding grants. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors extend their gratitude to Ariel Felner and Nathan Sturtevant for their contributions and assistance in the preliminary version of this work. We also note that Nathan Sturtevant independently derived the b-bound from the lb_C definition.

This work was supported by The Israel Science Foundation (ISF) grant #909/23 and by Israel's Ministry of Innovation, Science and Technology (MOST) grant #1001706842, in collaboration with Israel National Road Safety Authority and Netivei Israel.

Appendix A. Full derivation of bounds

This appendix elaborates on the full derivation process of all bounds presented in Section 3.

$$
\begin{aligned}
lb_{CU}(u,v) &\geq g_F(u) + g_B(v) + [\text{term } 3a] \\
&= g_F(u) + g_B(v) + h_F(u) - h_F(v) \\
&= f_F(u) + d_B(v) && (B_1) \\
&\geq fMin_F + dMin_B = B_1 && S' = \{[\text{term } 3a]\}
\end{aligned}
$$

$$
\begin{aligned}
lb_{CU}(u,v) &\geq g_F(u) + g_B(v) + [\text{term } 3b] \\
&= g_F(u) + g_B(v) + h_B(v) - h_B(u) \\
&= d_F(u) + f_B(v) && (B_2) \\
&\geq dMin_F + fMin_B = B_2 && S' = \{[\text{term } 3b]\}
\end{aligned}
$$

$$
\begin{aligned}
lb_C(u,v) &\geq g_F(u) + g_B(v) + [\text{term } 3c] \\
&= g_F(u) + g_B(v) + \epsilon && (B_3) \\
&\geq gMin_F + gMin_B + \epsilon = B_3 && S' = \{[\text{term } 3c]\}
\end{aligned}
$$

$$
\begin{aligned}
lb_{CU}(u,v) &\geq g_F(u) + g_B(v) + \frac{[\text{term } 3a] + [\text{term } 3b]}{2} \\
&= g_F(u) + g_B(v) + \frac{h_F(u) - h_F(v) + h_B(v) - h_B(u)}{2} \\
&= \frac{2g_F(u) + 2g_B(v) + h_F(u) - h_F(v) + h_B(v) - h_B(u)}{2} \\
&= \frac{f_F(u) + d_F(u) + f_B(v) + d_B(v)}{2} = \frac{b_F(u) + b_B(v)}{2} && (B_4) \\
&\geq \frac{bMin_F(u) + bMin_B(v)}{2} = B_4 && S' = \{[\text{term } 3a], [\text{term } 3b]\}
\end{aligned}
$$

$$
\begin{aligned}
lb_{CU}(u,v) &\geq g_F(u) + g_B(v) + \frac{[\text{term } 3a] + [\text{term } 3c]}{2} \\
&= g_F(u) + g_B(v) + \frac{h_F(u) - h_F(v) + \epsilon}{2} \\
&= \frac{2g_F(u) + 2g_B(v) + h_F(u) - h_F(v) + \epsilon}{2} \\
&= \frac{f_F(u) + g_F(u) + d_B(v) + g_B(v) + \epsilon}{2} && (B_5) \\
&\geq \frac{\{f,g\}Min_F + \{d,g\}Min_B + \epsilon}{2} = B_5 && S' = \{[\text{term } 3a], [\text{term } 3c]\}
\end{aligned}
$$

$$
\begin{aligned}
lb_{CU}(u,v) &\geq g_F(u) + g_B(v) + \frac{[\text{term } 3b] + [\text{term } 3c]}{2} \\
&= g_F(u) + g_B(v) + \frac{h_B(v) - h_B(u) + \epsilon}{2}
\end{aligned}
$$

$$= \frac{2g_F(u) + 2g_B(v) + h_B(v) - h_B(u) + \epsilon}{2}$$

$$= \frac{d_F(u) + g_F(u) + f_B(v) + g_B(v) + \epsilon}{2} \qquad (B_6)$$

$$\geq \frac{\{d,g\}Min_F + \{f,g\}Min_B + \epsilon}{2} = B_6 \qquad S' = \{[\text{term } 3b], [\text{term } 3c]\}$$

$$lb_{CU}(u,v) \geq g_F(u) + g_B(v) + \frac{[\text{term } 3a] + [\text{term } 3b] + [\text{term } 3c]}{3}$$

$$= g_F(u) + g_B(v) + \frac{h_F(u) - h_F(v) + h_B(v) - h_B(u) + \epsilon}{3}$$

$$= \frac{3g_F(u) + 3g_B(v) + h_F(u) - h_F(v) + h_B(v) - h_B(u) + \epsilon}{3}$$

$$= \frac{b_F(u) + g_F(u) + b_B(v) + g_B(v) + \epsilon}{3} \qquad (B_7)$$

$$\geq \frac{\{b,g\}Min_F + \{b,g\}Min_B + \epsilon}{3} = B_7 \qquad S' = \{[\text{term } 3a], [\text{term } 3b], [\text{term } 3c]\}$$

$$lb_{CU}(u,v) \geq g_F(u) + g_B(v) + [\text{term } 3d]$$

$$= g_F(u) + g_B(v) + h_F(v) - h_F(u)$$

$$= rf_F(u) + rd_B(v) \qquad (B_8)$$

$$\geq rfMin_F + rdMin_B = B_8 \qquad S' = \{[\text{term } 3d]\}$$

$$lb_{CU}(u,v) \geq g_F(u) + g_B(v) + [\text{term } 3e]$$

$$= g_F(u) + g_B(v) + h_B(u) - h_B(v)$$

$$= rd_F(u) + rf_B(v) \qquad (B_9)$$

$$\geq rdMin_F + rfMin_B = B_9 \qquad S' = \{[\text{term } 3e]\}$$

$$lb_{CU}(u,v) \geq g_F(u) + g_B(v) + \frac{[\text{term } 3a] + [\text{term } 3e]}{2}$$

$$= \frac{2g_F(u) + 2g_B(v) + h_F(u) - h_F(v) + h_B(u) - h_B(v)}{2}$$

$$= \frac{f_F(u) + rd_F(u) + rf_B(v) + d_B(v)}{2} \qquad (B_{10})$$

$$= \frac{\{f,rd\}Min_F + \{rf,d\}Min_B}{2} = B_{10} \qquad S' = \{[\text{term } 3a], [\text{term } 3e]\}$$

$$lb_{CU}(u,v) \geq g_F(u) + g_B(v) + \frac{[\text{term } 3d] + [\text{term } 3b]}{2}$$

$$= g_F(u) + g_B(v) + \frac{h_F(v) - h_F(u) + h_B(v) - h_B(u)}{2}$$

$$= \frac{2g_F(u) + 2g_B(v) + h_F(v) - h_F(u) + h_B(v) - h_B(u)}{2}$$

$$= \frac{rf_F(u) + d_F(u) + f_B(v) + rd_B(v)}{2} \qquad (B_{11})$$

$$\geq \frac{\{rf,d\}Min_F + \{f,rd\}Min_B}{2} = B_{11} \qquad S' = \{[\text{term } 3b], [\text{term } 3d]\}$$

$$lb_{CU}(u,v) \geq g_F(u) + g_B(v) + \frac{[\text{term } 3c] + [\text{term } 3d]}{2}$$

$$= g_F(u) + g_B(v) + \frac{h_F(v) - h_F(u) + \epsilon}{2}$$

$$= \frac{2g_F(u) + 2g_B(v) + h_F(v) - h_F(u) + \epsilon}{2}$$

$$= \frac{rf_F(u) + g_F(u) + rd_B(v) + g_B(v) + \epsilon}{2} \qquad (B_{12})$$

$$\geq \frac{\{rf,g\}Min_F + \{rd,g\}Min_B + \epsilon}{2} = B_{12} \qquad S' = \{[\text{term } 3c], [\text{term } 3d]\}$$

$$
\begin{aligned}
lb_{CU}(u,v) &\geq g_F(u) + g_B(v) + \frac{[\text{term } 3c] + [\text{term } 3e]}{2} \\
&= g_F(u) + g_B(v) + \frac{h_B(u) - h_B(v) + \epsilon}{2} \\
&= \frac{2g_F(u) + 2g_B(v) + h_B(u) - h_B(v) + \epsilon}{2} \\
&= \frac{rd_F(u) + g_F(u) + rf_B(v) + g_B(v) + \epsilon}{2} && (B_{13}) \\
&\geq \frac{\{rd, g\}Min_F + \{rf, g\}Min_B + \epsilon}{2} = B_{13} && S' = \{[\text{term } 3c], [\text{term } 3e]\}
\end{aligned}
$$

$$
\begin{aligned}
lb_{CU}(u,v) &\geq g_F(u) + g_B(v) + \frac{[\text{term } 3d] + [\text{term } 3e]}{2} \\
&= \frac{2g_F(u) + 2g_B(v) + h_F(v) - h_F(u) + h_B(u) - h_B(v)}{2} \\
&= \frac{rf_F(u) + rd_F(u) + rf_B(v) + rd_B(v)}{2} && (B_{14}) \\
&\geq \frac{\{rf, rd\}Min_F + \{rf, rd\}Min_B}{2} = B_{14} && S' = \{[\text{term } 3d], [\text{term } 3e]\}
\end{aligned}
$$

$$
\begin{aligned}
lb_{CU}(u,v) &\geq g_F(u) + g_B(v) + \frac{[\text{term } 3b] + [\text{term } 3c] + [\text{term } 3d]}{3} \\
&= g_F(u) + g_B(v) + \frac{h_F(u) - h_F(v) + h_B(u) - h_B(v) + \epsilon}{3} \\
&= \frac{3g_F(u) + 3g_B(v) + h_F(u) - h_F(v) + h_B(u) - h_B(v) + \epsilon}{3} \\
&= \frac{f_F(u) + rd_F(u) + g_F(u) + rf_B(v) + d_B(v) + g_B(v) + \epsilon}{3} && (B_{15}) \\
&\geq \frac{\{f, rd, g\}Min_F + \{rf, d, g\}Min_B + \epsilon}{3} = B_{15} && S' = \{[\text{term } 3a], [\text{term } 3c], [\text{term } 3e]\}
\end{aligned}
$$

$$
\begin{aligned}
lb_{CU}(u,v) &\geq g_F(u) + g_B(v) + \frac{[\text{term } 3b] + [\text{term } 3c] + [\text{term } 3d]}{3} \\
&= g_F(u) + g_B(v) + \frac{h_F(v) - h_F(u) + h_B(v) - h_B(u) + \epsilon}{3} \\
&= \frac{3g_F(u) + 3g_B(v) + h_F(v) - h_F(u) + h_B(v) - h_B(u) + \epsilon}{3} \\
&= \frac{rf_F(u) + d_F(u) + g_F(u) + f_B(v) + rd_B(v) + g_B(v) + \epsilon}{3} && (B_{16}) \\
&\geq \frac{\{rf, d, g\}Min_F + \{f, rd, g\}Min_B + \epsilon}{3} = B_{16} && S' = \{[\text{term } 3b], [\text{term } 3c], [\text{term } 3d]\}
\end{aligned}
$$

$$
\begin{aligned}
lb_{CU}(u,v) &\geq g_F(u) + g_B(v) + \frac{[\text{term } 3b] + [\text{term } 3c] + [\text{term } 3e]}{3} \\
&= g_F(u) + g_B(v) + \frac{h_F(v) - h_F(u) + h_B(u) - h_B(v) + \epsilon}{3} \\
&= \frac{3g_F(u) + 3g_B(v) + h_F(v) - h_F(u) + h_B(u) - h_B(v) + \epsilon}{3} \\
&= \frac{rf_F(u) + rd_F(u) + g_F(u) + rf_B(v) + rd_B(v) + g_B(v) + \epsilon}{3} && (B_{17}) \\
&\geq \frac{\{rf, rd, g\}Min_F + \{rf, rd, g\}Min_B + \epsilon}{3} = B_{17} && S' = \{[\text{term } 3b], [\text{term } 3c], [\text{term } 3e]\}
\end{aligned}
$$

Appendix B. Numerical examples for non-domination of bounds

Table B.14 contains examples of OPEN_F and OPEN_B such that each bound from B_1 to B_7 attains the maximum value. Each node is represented by a tuple $(g_D, h_D, h_{\overline{D}})$, which determines its contribution to the corresponding bound term. We also assume that $\epsilon = 1$ as in the tested domains.

Table B.14
Numerical Examples for Bound Non-Domination.

Bound	OPEN_F	OPEN_B	B_1	B_2	B_3	B_4	B_5	B_6	B_7
B_1	(2,0,0), (0,2,0)	(0,0,0), (0,0,0)	**2**	0	1	1	1.5	0.5	1
B_2	(0,0,0), (0,0,0)	(2,0,0), (0,2,0)	0	**2**	1	1	0.5	1.5	1
B_3	(0,0,0), (0,0,0)	(0,0,0), (0,0,0)	0	0	**1**	0	0.5	0.5	0.33
B_4	(0,2,0), (1,0,0)	(0,2,0), (1,0,0)	1	1	1	**2**	1.5	1.5	1.67
B_5	(1,0,0), (0,2,0)	(0,0,0), (0,0,0)	1	0	1	1	**1.5**	0.5	1
B_6	(0,0,0), (0,0,0)	(1,0,0), (0,2,0)	0	1	1	1	0.5	**1.5**	1
B_7	(1,0,0), (0,3,0)	(0,1,0), (0,1,0)	1	1	1	1.5	1.5	1	**1.67**

Appendix C. Full experimental tables

This appendix presents the complete results of our experiments, organized by domain. Each table provides the performance of all algorithms and side-selection strategies across the heuristics discussed in Section 6.1. Specifically, Table C.15 contains results for the Pancake Puzzle domain, Table C.16 corresponds to the Towers of Hanoi (ToH) domain, Table C.17 displays the results for the Sliding Tile Puzzle (STP), Table C.18 shows the results on grid-based pathfinding, and Table C.19 includes results for road networks.

Table C.15
Results for Pancake.

Algorithm	GAP		GAP-1		GAP-2		GAP-3		GAP-4		GAP-5	
	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$
A*	**72**	46	8,423	8,400	351K	348K	6,838K	6,834K	#46		#25	
rA*	77	52	7,077	6,854	350K	349K	6,906K	6,830K	#47		#26	
NBS	148	66	6,477	6,220	123K	122K	699K	699K	1,433K	1,433K	1,623K	1,623K
DVCBS	209	47	4,885	4,635	85K	84K	492K	479K	977K	899K	1,137K	1,039K
MVC(GMX)		**39**		4,547		82K		452K	#39		#16	
$\text{MVC}(\text{GMX}_\text{C})$		37		297		6,711		59K	N\A		N\A	
$\text{MVC}(\text{GMX}_\text{CU})$		37		297		6,711		59K	N\A		N\A	
BAE*(a)	90	66	654	552	15K	11K	132K	113K	469K	465K	1,116K	1,087K
BAE*(p)	88	64	730	621	18K	13K	154K	135K	480K	476K	1,116K	1,087K
BAE*(FBI)	75	**39**	**427**	**317**	11K	7,191	92K	77K	362K	343K	869K	834K
TB_1(a)	136	86	1,406	1,330	78K	78K	2,042K	1,879K	5,581K	5,492K	10,232K	10,133K
TB_1(p)	150	100	6,766	6,712	366K	365K	3,691K	3,028K	8,128K	7,708K	11,371K	11,063K
TB_1(FBI)	#21		#9		#48		#49		5,670K	5,134K	8,047K	7,521K
TB_2(a)	149	100	1,640	1,446	81K	79K	1,758K	1,755K	6,178K	6,036K	11,337K	11,020K
TB_2(p)	154	101	6,361	6,325	378K	364K	3,099K	2,780K	7,339K	7,207K	13,428K	12,363K
TB_2(FBI)	#24		#8		#49		#47		#47		8,864K	7,892K
TB_3(a)	484K	120K	657K	494K	1,508K	1,508K	1,623K	1,623K	1,623K	1,623K	1,624K	1,623K
TB_3(p)	488K	124K	649K	486K	1,052K	1,052K	1,089K	1,089K	1,291K	1,290K	1,535K	1,535K
TB_3(FBI)	1,053K	240K	1,053K	639K	1,054K	1,053K	1,055K	1,053K	1,058K	1,053K	1,065K	1,053K
TB_4(a)	90	66	654	552	15K	11K	132K	113K	469K	465K	1,116K	1,087K
TB_4(p)	88	64	730	621	18K	13K	154K	135K	480K	476K	1,116K	1,087K
TB_4(FBI)	75	**39**	**427**	**317**	11K	7,191	92K	77K	362K	343K	869K	834K
TB_5(a)	27K	16K	60K	48K	150K	133K	353K	336K	674K	634K	1,006K	972K
TB_5(p)	26K	16K	59K	45K	176K	134K	400K	383K	742K	696K	1,072K	1,033K
TB_5(FBI)	#48		#48		#45		#47		#47		#47	
TB_6(a)	26K	16K	52K	46K	153K	142K	369K	341K	741K	658K	1,137K	1,010K
TB_6(p)	25K	16K	55K	43K	161K	147K	383K	359K	867K	785K	1,251K	1,107K
TB_6(FBI)	25K	12K	43K	31K	126K	101K	305K	243K	600K	489K	958K	786K
TB_7(a)	2,724	2,073	6,636	5,861	35K	33K	114K	106K	301K	282K	596K	555K
TB_7(p)	2,862	2,209	6,173	5,397	36K	34K	117K	109K	301K	282K	595K	554K
TB_7(FBI)	3,704	1,633	6,705	4,793	31K	25K	101K	86K	257K	228K	498K	456K
TB_8(a)	#16		#12		#24		#22		#29		#34	
TB_8(p)	#18		#14		#24		#21		#27		#32	
TB_9(a)	#20		#11		#20		#21		#25		#32	
TB_9(p)	#23		#12		#24		#22		#25		#31	
TB_{10}(a)	2,582K	840K	3,147K	2,248K	9,083K	9,022K	12,472K	12,471K	13,772K	13,772K	14,000K	14,000K
TB_{10}(p)	2,556K	797K	3,855K	2,913K	9,197K	9,109K	13,202K	13,177K	16,274K	16,231K	17,139K	17,095K
TB_{11}(a)	2,499K	848K	3,284K	2,195K	9,079K	9,003K	11,845K	11,845K	13,007K	13,004K	#49	
TB_{11}(p)	2,467K	860K	4,054K	2,826K	9,001K	8,902K	12,033K	12,033K	14,538K	14,538K	16,840K	16,680K

Table C.15 (*continued*)

Algorithm	GAP		GAP-1		GAP-2		GAP-3		GAP-4		GAP-5	
	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$
TB_{12}(a)	16,002K	7,256K	#49		#49		21,865K	21,809K	17,511K	17,351K	#49	
TB_{12}(p)	16,662K	8,065K	#49		#49		30,727K	30,387K	23,440K	23,439K	17,418K	17,173K
TB_{13}(a)	#49		#47		#49		21,964K	21,221K	17,450K	16,838K	12,514K	12,514K
TB_{13}(p)	#49		#47		#49		30,726K	29,923K	23,383K	23,136K	16,219K	16,219K
TB_{14}(a)	#16		#21		#35		#39		#43		#42	
TB_{14}(p)	#16		#21		#35		#39		#42		#42	
TB_{15}(a)	1,884K	564K	2,264K	1,679K	3,873K	3,851K	4,074K	4,060K	4,257K	4,250K	4,395K	4,253K
TB_{15}(p)	1,905K	568K	2,390K	1,805K	4,428K	4,381K	5,052K	4,998K	5,412K	5,396K	5,041K	4,866K
TB_{16}(a)	1,901K	567K	1,935K	1,308K	3,889K	3,860K	4,121K	4,120K	4,487K	4,338K	4,480K	4,406K
TB_{16}(p)	1,922K	576K	2,368K	1,776K	4,435K	4,326K	5,361K	5,250K	5,773K	5,577K	5,172K	5,091K
TB_{17}(a)	#48		#47		#45		33,655K	33,211K	27,239K	27,239K	#49	
TB_{17}(p)	#48		#47		#45		38,726K	37,871K	33,728K	33,649K	#49	
$DBBS_b$(a)	114	57	688	458	10K	8,801	77K	70K	277K	272K	653K	640K
$DBBS_b$(p)	152	42	610	326	9,143	7,076	72K	64K	258K	248K	592K	574K
$DBBS_b$(FBI)	103	42	601	326	9,389	7,076	71K	64K	259K	247K	603K	569K
$DBBS_{ss}$(a)	113	57	634	457	11K	9,437	80K	78K	269K	267K	526K	502K
$DBBS_{ss}$(p)	146	43	542	329	9,140	7,658	73K	68K	239K	235K	481K	454K
$DBBS_{ss}$(FBI)	104	43	517	353	8,545	7,678	69K	67K	232K	230K	489K	452K
$DBBS_{max}$(a)	114	57	688	458	10K	8,801	77K	70K	277K	272K	653K	640K
$DBBS_{max}$(p)	152	42	610	326	9,143	7,076	72K	64K	258K	248K	592K	574K
$DBBS_{max}$(FBI)	103	42	601	326	9,389	7,076	71K	64K	259K	247K	603K	569K
$DBBS_b^{all}$(a)	114	57	688	458	10K	8,801	77K	69K	274K	269K	639K	626K
$DBBS_b^{all}$(p)	152	42	610	326	9,140	**7,073**	72K	64K	256K	246K	579K	559K
$DBBS_b^{all}$(FBI)	103	42	601	326	9,386	**7,073**	71K	**63K**	258K	246K	603K	564K
$DBBS_{ss}^{all}$(a)	113	57	628	450	10K	9,087	75K	73K	250K	246K	515K	485K
$DBBS_{ss}^{all}$(p)	145	43	522	327	8,634	7,500	69K	65K	233K	222K	**474K**	**433K**
$DBBS_{ss}^{all}$(FBI)	104	43	523	353	**8,410**	7,562	**66K**	64K	**223K**	**219K**	475K	435K
$DBBS_{max}^{all}$(a)	114	57	688	458	10K	8,801	77K	69K	271K	266K	523K	508K
$DBBS_{max}^{all}$(p)	152	42	610	326	9,140	**7,073**	72K	64K	252K	242K	485K	463K
$DBBS_{max}^{all}$(FBI)	103	42	601	326	9,386	**7,073**	71K	**63K**	255K	243K	493K	463K

Table C.16
Results for ToH.

Algorithm	(10+2)		(8+4)		(6+6)	
	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$	$\leq C^*$	$< C^*$
A*	276K	276K	1,926K	1,925K	3,268K	3,239K
rA*	123K	123K	622K	620K	1,064K	1,033K
NBS	230K	230K	647K	645K	682K	662K
DVCBS	232K	232K	619K	609K	663K	635K
MVC(GMX)		123K		557K		624K
MVC(GMX_C)		41K		174K		363K
MVC(GMX_{CU})		41K		174K		363K
BAE*(a)	47K	46K	187K	186K	383K	382K
BAE*(p)	46K	45K	185K	182K	375K	374K
BAE*(FBI)	**45K**	**43K**	**181K**	**178K**	**368K**	**366K**
TB_1(a)	54K	53K	204K	204K	424K	423K
TB_1(p)	50K	50K	200K	200K	423K	421K
TB_1(FBI)	**45K**	45K	190K	190K	404K	402K
TB_2(a)	69K	69K	240K	240K	435K	435K
TB_2(p)	60K	60K	221K	221K	408K	407K
TB_2(FBI)	56K	56K	212K	210K	396K	394K
TB_3(a)	648K	622K	662K	660K	683K	664K
TB_3(p)	623K	599K	645K	643K	672K	652K
TB_3(FBI)	629K	603K	635K	628K	656K	629K
TB_4(a)	47K	46K	187K	186K	383K	382K
TB_4(p)	46K	45K	185K	182K	375K	374K
TB_4(FBI)	**45K**	**43K**	**181K**	**178K**	**368K**	**366K**
TB_5(a)	263K	262K	386K	382K	515K	509K
TB_5(p)	253K	251K	385K	379K	507K	500K
TB_5(FBI)	259K	251K	384K	371K	504K	491K
TB_6(a)	282K	277K	412K	406K	513K	512K
TB_6(p)	264K	257K	394K	390K	500K	497K
TB_6(FBI)	266K	254K	394K	382K	495K	489K
TB_7(a)	170K	165K	315K	308K	458K	455K
TB_7(p)	165K	158K	309K	303K	451K	446K
TB_7(FBI)	168K	159K	310K	300K	448K	440K
TB_8(a)	1,880K	1,809K	1,588K	1,588K	1,285K	1,282K
TB_8(p)	1,708K	1,672K	1,437K	1,437K	1,205K	1,200K
TB_9(a)	1,860K	1,840K	1,471K	1,471K	1,234K	1,231K
TB_9(p)	1,805K	1,783K	1,447K	1,447K	1,203K	1,199K
TB_{10}(a)	729K	728K	717K	717K	742K	742K
TB_{10}(p)	701K	699K	706K	706K	739K	737K
TB_{11}(a)	765K	751K	788K	788K	770K	770K
TB_{11}(p)	683K	675K	740K	740K	744K	743K
TB_{12}(a)	1,252K	1,189K	1,094K	1,094K	937K	930K
TB_{12}(p)	1,185K	1,133K	1,050K	1,050K	910K	908K
TB_{13}(a)	1,218K	1,189K	1,041K	1,039K	909K	904K
TB_{13}(p)	1,221K	1,185K	1,043K	1,043K	905K	900K
TB_{14}(a)	1,749K	1,671K	1,484K	1,480K	1,211K	1,208K
TB_{14}(p)	1,674K	1,602K	1,403K	1,401K	1,168K	1,167K
TB_{15}(a)	693K	688K	684K	678K	700K	698K
TB_{15}(p)	683K	675K	677K	673K	696K	694K
TB_{16}(a)	706K	677K	729K	727K	714K	713K
TB_{16}(p)	663K	643K	700K	698K	698K	695K
TB_{17}(a)	1,394K	1,314K	1,199K	1,194K	1,003K	1,000K
TB_{17}(p)	1,351K	1,280K	1,165K	1,161K	992K	988K
$DBBS_b$(a)	48K	46K	189K	186K	383K	379K
$DBBS_b$(p)	**45K**	**43K**	**181K**	**178K**	371K	**366K**
$DBBS_b$(FBI)	**45K**	**43K**	**181K**	**178K**	370K	**366K**
$DBBS_{ss}$(a)	51K	49K	197K	196K	399K	398K
$DBBS_{ss}$(p)	46K	44K	182K	179K	371K	369K
$DBBS_{ss}$(FBI)	48K	47K	194K	192K	395K	393K
$DBBS_{max}$(a)	48K	46K	192K	189K	390K	387K
$DBBS_{max}$(p)	**45K**	**43K**	189K	186K	387K	380K
$DBBS_{max}$(FBI)	**45K**	**43K**	189K	186K	381K	377K
$DBBS^{all}_b$(a)	48K	46K	189K	186K	383K	379K
$DBBS^{all}_b$(p)	**45K**	**43K**	**181K**	**178K**	371K	**366K**
$DBBS^{all}_b$(FBI)	**45K**	**43K**	**181K**	**178K**	370K	**366K**
$DBBS^{all}_{ss}$(a)	52K	50K	198K	197K	403K	402K
$DBBS^{all}_{ss}$(p)	46K	44K	183K	180K	376K	374K
$DBBS^{all}_{ss}$(FBI)	50K	48K	195K	194K	398K	397K
$DBBS^{all}_{max}$(a)	48K	46K	192K	189K	390K	387K
$DBBS^{all}_{max}$(p)	**45K**	**43K**	189K	186K	385K	379K
$DBBS^{all}_{max}$(FBI)	**45K**	**43K**	189K	186K	381K	377K

Table C.17
Results for STP.

Algorithm	$\leq C^*$	$< C^*$
A*	15,550K	14,700K
rA*	11,144K	10,956K
NBS	12,556K	11,739K
DVCBS	10,930K	10,720K
MVC(GMX)		9,267K
MVC(GMX_C)		1,308K
MVC(GMX_{CU})		1,308K
BAE*(a)	2,707K	2,700K
BAE*(p)	2,837K	2,829K
BAE*(FBI)	2,258K	2,211K
TB_1(a)	13,404K	12,953K
TB_1(p)	16,087K	15,863K
TB_1(FBI)	92,651K	92,303K
TB_2(a)	10,815K	10,677K
TB_2(p)	12,253K	12,082K
TB_2(FBI)	#64	
TB_3(a)	#60	
TB_3(p)	#60	
TB_3(FBI)	#54	
TB_4(a)	2,707K	2,700K
TB_4(p)	2,837K	2,829K
TB_4(FBI)	2,258K	2,211K
TB_5(a)	#95	
TB_5(p)	#95	
TB_5(FBI)	#95	
TB_6(a)	#96	
TB_6(p)	#96	
TB_6(FBI)	#96	
TB_7(a)	13,448K	13,275K
TB_7(p)	12,880K	12,707K
TB_7(FBI)	12,973K	12,692K
TB_8(a)	#0	
TB_8(p)	#0	
TB_9(a)	#0	
TB_9(p)	#0	
TB_{10}(a)	#19	
TB_{10}(p)	#19	
TB_{11}(a)	#23	
TB_{11}(p)	#24	
TB_{12}(a)	#8	
TB_{12}(p)	#8	
TB_{13}(a)	#8	
TB_{13}(p)	#8	
TB_{14}(a)	#0	
TB_{14}(p)	#0	
TB_{15}(a)	#34	
TB_{15}(p)	#34	
TB_{16}(a)	#36	
TB_{16}(p)	#39	
TB_{17}(a)	#5	
TB_{17}(p)	#6	
$DBBS_b$(a)	2,262K	1,701K
$DBBS_b$(p)	2,011K	**1,314K**
$DBBS_b$(FBI)	**1,977K**	**1,314K**
$DBBS_{ss}$(a)	2,306K	1,954K
$DBBS_{ss}$(p)	2,063K	1,691K
$DBBS_{ss}$(FBI)	2,027K	1,677K
$DBBS_{max}$(a)	2,646K	2,041K
$DBBS_{max}$(p)	2,216K	1,491K
$DBBS_{max}$(FBI)	2,318K	1,723K
$DBBS^{all}_b$(a)	2,262K	1,701K
$DBBS^{all}_b$(p)	2,011K	**1,314K**
$DBBS^{all}_b$(FBI)	**1,977K**	**1,314K**
$DBBS^{all}_{ss}$(a)	2,292K	1,935K
$DBBS^{all}_{ss}$(p)	2,057K	1,668K
$DBBS^{all}_{ss}$(FBI)	2,054K	1,705K
$DBBS^{all}_{max}$(a)	2,646K	2,041K
$DBBS^{all}_{max}$(p)	2,216K	1,491K
$DBBS^{all}_{max}$(FBI)	2,318K	1,723K

Table C.18
Results for Grid.

Algorithm	$\leq C^*$	$< C^*$
A*	5,406	5,322
rA*	5,342	5,267
NBS	6,873	6,556
DVCBS	5,545	5,151
MVC(GMX)		4,290
MVC(GMX_C)		N\A
MVC(GMX_{CU})		N\A
BAE*(a)	6,718	6,668
BAE*(p)	5,996	5,862
BAE*(FBI)	6,207	6,161
TB_1(a)	7,081	6,940
TB_1(p)	5,823	5,461
TB_1(FBI)	6,652	6,481
TB_2(a)	7,150	7,018
TB_2(p)	5,861	5,465
TB_2(FBI)	6,658	6,471
TB_3(a)	10,275	8,654
TB_3(p)	9,514	7,739
TB_3(FBI)	9,528	7,764
TB_4(a)	6,706	6,483
TB_4(p)	5,986	5,634
TB_4(FBI)	6,197	5,889
TB_5(a)	9,213	7,955
TB_5(p)	8,521	7,128
TB_5(FBI)	8,604	7,188
TB_6(a)	9,221	8,021
TB_6(p)	8,567	7,201
TB_6(FBI)	8,605	7,221
TB_7(a)	8,562	7,563
TB_7(p)	7,939	6,806
TB_7(FBI)	8,172	6,909
TB_8(a)	11,835	9,735
TB_8(p)	10,606	8,440
TB_9(a)	12,005	9,751
TB_9(p)	10,529	8,293
TB_{10}(a)	10,713	8,896
TB_{10}(p)	9,478	7,663
TB_{11}(a)	10,599	8,922
TB_{11}(p)	9,526	7,676
TB_{12}(a)	11,108	9,235
TB_{12}(p)	10,299	8,292
TB_{13}(a)	11,147	9,214
TB_{13}(p)	10,348	8,274
TB_{14}(a)	11,647	9,566
TB_{14}(p)	10,805	8,548
TB_{15}(a)	10,446	8,744
TB_{15}(p)	9,696	7,903
TB_{16}(a)	10,411	8,787
TB_{16}(p)	9,653	7,867
TB_{17}(a)	11,267	9,313
TB_{17}(p)	10,426	8,312
$DBBS_b$(a)	6,105	5,829
$DBBS_b$(p)	5,690	5,396
$DBBS_b$(FBI)	**5,308**	4,945
$DBBS_{ss}$(a)	6,092	5,794
$DBBS_{ss}$(p)	5,660	5,384
$DBBS_{ss}$(FBI)	5,374	**4,935**
$DBBS_{max}$(a)	6,261	5,972
$DBBS_{max}$(p)	5,691	5,421
$DBBS_{max}$(FBI)	5,592	5,294
$DBBS_b^{all}$(a)	6,104	5,827
$DBBS_b^{all}$(p)	5,688	5,394
$DBBS_b^{all}$(FBI)	**5,308**	4,945
$DBBS_{ss}^{all}$(a)	6,094	5,797
$DBBS_{ss}^{all}$(p)	5,605	5,327
$DBBS_{ss}^{all}$(FBI)	5,595	5,181
$DBBS_{max}^{all}$(a)	6,163	5,829
$DBBS_{max}^{all}$(p)	5,608	5,327
$DBBS_{max}^{all}$(FBI)	5,677	5,384

Table C.19
Results for Road.

Algorithm	$\leq C^*$	$< C^*$
A*	127K	127K
rA*	126K	126K
NBS	96K	96K
DVCBS	126K	126K
MVC(GMX)		78K
MVC(GMX_C)		N\A
MVC(GMX_{CU})		N\A
BAE*(a)	67K	67K
BAE*(p)	**58K**	**58K**
BAE*(FBI)	158K	158K
TB_1(a)	70K	70K
TB_1(p)	61K	61K
TB_1(FBI)	127K	127K
TB_2(a)	72K	72K
TB_2(p)	62K	62K
TB_2(FBI)	217K	217K
TB_3(a)	119K	119K
TB_3(p)	106K	106K
TB_3(FBI)	110K	110K
TB_4(a)	67K	67K
TB_4(p)	**58K**	**58K**
TB_4(FBI)	158K	158K
TB_5(a)	96K	96K
TB_5(p)	84K	84K
TB_5(FBI)	176K	176K
TB_6(a)	96K	96K
TB_6(p)	85K	85K
TB_6(FBI)	220K	220K
TB_7(a)	86K	86K
TB_7(p)	75K	75K
TB_7(FBI)	184K	184K
TB_8(a)	163K	163K
TB_8(p)	147K	147K
TB_9(a)	162K	162K
TB_9(p)	147K	147K
TB_{10}(a)	124K	124K
TB_{10}(p)	111K	111K
TB_{11}(a)	125K	125K
TB_{11}(p)	112K	112K
TB_{12}(a)	141K	141K
TB_{12}(p)	125K	125K
TB_{13}(a)	140K	140K
TB_{13}(p)	125K	125K
TB_{14}(a)	156K	156K
TB_{14}(p)	140K	140K
TB_{15}(a)	121K	121K
TB_{15}(p)	108K	108K
TB_{16}(a)	121K	121K
TB_{16}(p)	109K	109K
TB_{17}(a)	145K	145K
TB_{17}(p)	129K	129K

Data availability

A link to the Github Repository containing the code and instructions for reproducing the results is included in the paper.

References

[1] V. Alcázar, The consistent case in bidirectional search and a bucket-to-bucket algorithm as a middle ground between front-to-end and front-to-front, in: ICAPS, AAAI Press, 2021, pp. 7–15.
[2] V. Alcázar, P.J. Riddle, M. Barley, A unifying view on individual bounds and heuristic inaccuracies in bidirectional search, in: AAAI, AAAI Press, 2020, pp. 2327–2334.
[3] Y. Boykov, V. Kolmogorov, An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision, IEEE Trans. Pattern Anal. Mach. Intell. 26 (2004) 1124–1137.
[4] J. Chen, R.C. Holte, S. Zilles, N.R. Sturtevant, Front-to-end bidirectional heuristic search with near-optimal node expansions, in: IJCAI, 2017, pp. 489–495, ijcai.org/.
[5] R. Dechter, J. Pearl, Generalized best-first search strategies and the optimality of A*, J. ACM 32 (1985) 505–536.
[6] C. Demetrescu, A.V. Goldberg, D.S. Johnson, in: The Shortest Path Problem: Ninth DIMACS Implementation Challenge, vol. 74, American Mathematical Soc., 2009.
[7] J. Eckerle, J. Chen, N.R. Sturtevant, S. Zilles, R.C. Holte, Sufficient conditions for node expansion in bidirectional heuristic search, in: ICAPS, 2017, pp. 79–87.
[8] A. Felner, R.E. Korf, R. Meshulam, R.C. Holte, Compressing pattern databases, in: National Conference on Artificial Intelligence (AAAI-04), 2004, pp. 638–643.
[9] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (1968) 100–107.
[10] M. Helmert, Landmark heuristics for the pancake problem, in: SoCS, AAAI Press, 2010, pp. 109–110.
[11] H. Kaindl, G. Kainz, Bidirectional heuristic search reconsidered, J. Artif. Intell. Res. 7 (1997) 283–317.
[12] D. König, Graphok es matrixok (Graphs and matrices) (Hungarian), Mat. Fiz. Lapok 38 (1931) 116–119.
[13] R.E. Korf, Depth-first iterative-deepening: an optimal admissible tree search, Artif. Intell. 27 (1985) 97–109.
[14] J.B.H. Kwa, BS*: an admissible bidirectional staged heuristic search algorithm, Artif. Intell. 38 (1989) 95–109.
[15] C.H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Courier Corporation, 1998.
[16] I. Pohl, Bi-directional search, in: B. Meltzer, D. Michie (Eds.), Machine Intelligence, vol. 6, Edinburgh University Press, 1971, pp. 127–140.
[17] S.K. Sadhukhan, A new approach to bidirectional heuristic search using error functions, in: CSI, 2012, pp. 239–243.
[18] E.C. Sewell, S.H. Jacobson, Dynamically improved bounds bidirectional search, Artif. Intell. 291 (2021) 103405.
[19] E. Shaham, A. Felner, J. Chen, N.R. Sturtevant, The minimal set of states that must be expanded in a front-to-end bidirectional search, in: SoCS, 2017, pp. 82–90.
[20] E. Shaham, A. Felner, N.R. Sturtevant, J.S. Rosenschein, Minimizing node expansions in bidirectional search with consistent heuristics, in: SoCS, AAAI Press, 2018, pp. 81–98.
[21] S.S. Shperberg, S. Danishevski, A. Felner, N.R. Sturtevant, Iterative-deepening bidirectional heuristic search with restricted memory, in: ICAPS, AAAI Press, 2021, pp. 331–339.
[22] S.S. Shperberg, A. Felner, S.E. Shimony, N.R. Sturtevant, A. Hayoun, Improving bidirectional heuristic search by bounds propagation, in: SoCS, 2019, pp. 106–114.
[23] S.S. Shperberg, A. Felner, N.R. Sturtevant, S.E. Shimony, A. Hayoun, Enriching non-parametric bidirectional search algorithms, in: AAAI, AAAI Press, 2019, pp. 2379–2386.
[24] L. Siag, S.S. Shperberg, BiHS Consistent F2E Repo, https://github.com/SPL-BGU/BiHS-Consistent-F2E, 2024.
[25] L. Siag, S.S. Shperberg, A. Felner, N.R. Sturtevant, Front-to-end bidirectional heuristic search with consistent heuristics: enumerating and evaluating algorithms and bounds, in: IJCAI, 2023, pp. 5631–5638, ijcai.org/.
[26] N.R. Sturtevant, Benchmarks for grid-based pathfinding, IEEE Trans. Comput. Intell. AI Games 4 (2012) 144–148.
[27] N.R. Sturtevant, S.S. Shperberg, A. Felner, J. Chen, Predicting the effectiveness of bidirectional heuristic search, in: ICAPS, AAAI Press, 2020, pp. 281–290.