Enriching Non-Parametric Bidirectional Search Algorithms

Shahaf S. Shperberg
CS Department
Ben-Gurion University
Be'er-Sheva, Israel
shperbsh@post.bgu.ac.il

Ariel Felner
ISE Department
Ben-Gurion University
Be'er-Sheva, Israel
felner@bgu.ac.il

Nathan R. Sturtevant
CS Department
University of Alberta
Canada
sturtevant@cs.du.edu

Solomon E. Shimony
Avi Hayoun
CS Department
Ben-Gurion University
Be'er-Sheva, Israel
shimony@cs.bgu.ac.il
hyounav@cs.bgu.ac.il

Abstract

NBS is a non-parametric bidirectional search algorithm proven to expand at most twice the number of node expansions required to verify the optimality of a solution. We introduce new variants of NBS that are aimed at finding all optimal solutions. We then introduce an algorithmic framework that includes NBS as a special case. Finally, we introduce DVCBS, a new algorithm in this framework that aims to further reduce the number of expansions. Unlike NBS, DVCBS does not have any worst-case bound guarantees, but in practice it outperforms NBS in verifying the optimality of solutions.

1 Introduction and Overview

Given a graph G, the shortest-path problem is to find the least-cost path from state s to state g in G. Bidirectional heuristic search algorithms (denoted henceforth by Bi-HS) interleave two separate searches, a search forward from s and a search backward from g. Recent research (Eckerle et al. 2017) defined conditions on the node expansions required by Bi-HS algorithms to guarantee solutions optimality. Following work reformulated these conditions as a *must-expand graph* (G_{MX}), showing that the *Minimum Vertex Cover* (MVC) of G_{MX} corresponds to the minimal number of expansions (Chen et al. 2017) required to prove optimality. Finally, Shaham et al. (2017; 2018) studied the G_{MX} structure and its extension, G_{MX_ϵ}, that exploits knowledge of the minimal edge cost (ϵ), to characterize properties of the MVC.

Bi-HS algorithms can be classified as *parametric* or as *non-parametric*. Two parametric algorithms were recently developed. *Fractional MM* (fMM(p)) (Shaham et al. 2017) generalizes the MM algorithm (Holte et al. 2017) by controlling the fraction p of the optimal path at which the forward and backward frontiers meet. There exists an optimal fraction p^* for which fMM(p^*) will expand exactly an MVC of G_{MX}, but p^* is not known a priori. Another parametric algorithm is GBFHS (Barley et al. 2018), which iteratively increases the depth of the search. It is parametric in a predefined *split function* that determines how deep to search on each side at each iteration. GBFHS with an optimal split function also converges to an MVC of G_{MX}. However, such a split function is not known a priori. Without knowledge of the optimal parameter values, both algorithms may expand many more nodes than an MVC of G_{MX}.

In this paper we focus on non-parametric Bi-HS algorithms. NBS (Chen et al. 2017) is a robust state-of-the-art non-parametric algorithm that computes a *vertex cover* (VC) of G_{MX} whose size is at most 2|MVC|. We enrich this line of research and introduce new settings and new algorithms that aim to find a VC of G_{MX}. In particular, we make the following contributions:

(1) We describe and motivate the problem of finding *all* optimal solutions, and introduce two new versions of G_{MX} (with/without ϵ) that are suited for such settings. This results in four different problem settings, each with its own G_{MX}.
(2) We introduce a 2-level framework for non-parametric Bi-HS algorithms and reformulate NBS as a special case.
(3) Utilizing our framework, we adapt NBS to the four settings, while maintaining the 2|MVC| guarantee.
(4) We introduce a new algorithm ***D**ynamic **V**ertex **C**over **B**idirectional **S**earch* (DVCBS). It uses the same high-level framework we developed, but unlike NBS, always tries to expand a VC of a dynamic G_{MX} graph which is also introduced. Here too, four versions are possible.
(5) Our experimental results show that the new variants of NBS, as well as DVCBS, outperform previous variants of NBS for finding both the *first* and *all* optimal solutions, expanding significantly fewer nodes in many cases.

2 Definitions and Background

Let $d(x, y)$ denote the shortest distance between x and y, $C^* = d(s, g)$, and let f_F, g_F and h_F indicate f-, g-, and h-costs in the forward search, and likewise f_B, g_B and h_B in the backward search. The *forward heuristic* h_F is *admissible* iff $h_F(u) \leq d(u, g)$ for every state $u \in G$ and is *consistent* iff $h_F(u) \leq d(u, u') + h_F(u')$ for all $u, u' \in G$. The *backward heuristic* h_B is defined analogously. *Front-to-end* Bi-HS algorithms use these two heuristic functions and in this paper we assume that both are admissible and consistent. *Front-to-front* Bi-HS algorithms use heuristics between pairs of states on opposite frontiers, and are outside the focus of this paper; see Holte et al. (2017) for a survey.

2.1 Guaranteeing Solution Optimality

Unidirectional search algorithms must expand all nodes n with $f(n) < C^*$ in order to guarantee the optimality of solutions (Dechter and Pearl 1985).

Eckerle et al. (2017) generalized this to Bi-HS by examining pairs of nodes $\langle u, v\rangle$ such that u is in the forward frontier and v is in the backward frontier. They defined conditions for when such pairs should be expanded:

1. $f_F(u) < C^*$
2. $f_B(v) < C^*$
3. $g_F(u) + g_B(v) < C^*$

If u and v meet the three conditions, then to guarantee solution optimality every algorithm must expand at least one of u or v in order to ensure that there is no path from s to g passing through u and v of cost $< C^*$.

Definition 1. *For each pair of states* (u, v) *let*
$$lb(u, v) = \max\{f_F(u), f_B(v), g_F(u) + g_B(v)\}$$

In Bi-HS, a pair of states $\langle u, v\rangle$ is called a *must-expand pair* (MEP) if $lb(u, v) < C^*$. The MEP definition is equivalent to the above conditions; for each MEP only *one* of u or v *must* be expanded. In the special case of unidirectional search, algorithms expand all the nodes with $f_F < C^*$, which is equivalent to expanding the forward node of every MEP. Bi-HS algorithms may expand nodes from either side, potentially covering all the MEPs with fewer expansions.

Shaham et al. (2018) generalized the three conditions to handle the case where a lower bound ϵ on the edge costs is available. In unit edge-cost domains $\epsilon = 1$, while in other domains one might iterate over all action costs and set ϵ to their minimum. We denote this case by ϵ-case, as opposed to the base-case, where no knowledge of ϵ is available. For ϵ-case, Condition 3 is changed to:

3. $g_F(u) + g_B(v) + \epsilon < C^*$

Consequently, the lower bound is changed to:
$$lb(u, v) = \max\{f_F(u), f_B(v), g_F(u) + g_B(v) + \epsilon\}$$
and an MEP is defined according to the new lb.

2.2 The Must-Expand Graph (G_{MX})

The problem of selecting the minimal set of nodes that cover all MEPs can be restated as finding an MVC on the *must-expand graph* (Chen et al. 2017).

Definition 2. *The Must-Expand Graph* (G_{MX}) *of a problem instance is an undirected, unweighted bipartite graph. For each state* $u \in G$ *there is a left vertex* u_F *and a right vertex* u_B. G_{MX} *has an edge between a left vertex* u_F *and a right vertex* v_B *if and only if* (u, v) *is an* MEP.

It follows that Bi-HS algorithms must expand a vertex cover (VC) of the induced G_{MX} when solving a problem instance. The MVC is thus a lower bound on the number of expansions. Another version of G_{MX}, denoted by G_{MX_ϵ}, can be constructed for ϵ-case (Shaham et al. 2018).

Figure 1 illustrates different versions of G_{MX} for the problem instance in Figure 1(a), in which $C^* = 3$. Figure 1(b) shows the corresponding G_{MX}. The left (right) vertices are ordered by increasing (decreasing) g_F-costs (g_B-costs). Additionally, vertices with identical g_F (or g_B) are merged into a single *weighted vertex*, denoted as a *cluster*. For example the *cluster* with $g_F = 1$ includes both A and X and its weight is 2. Similarly, an edge that connects clusters represents all possible edges between them (the product of their weights),

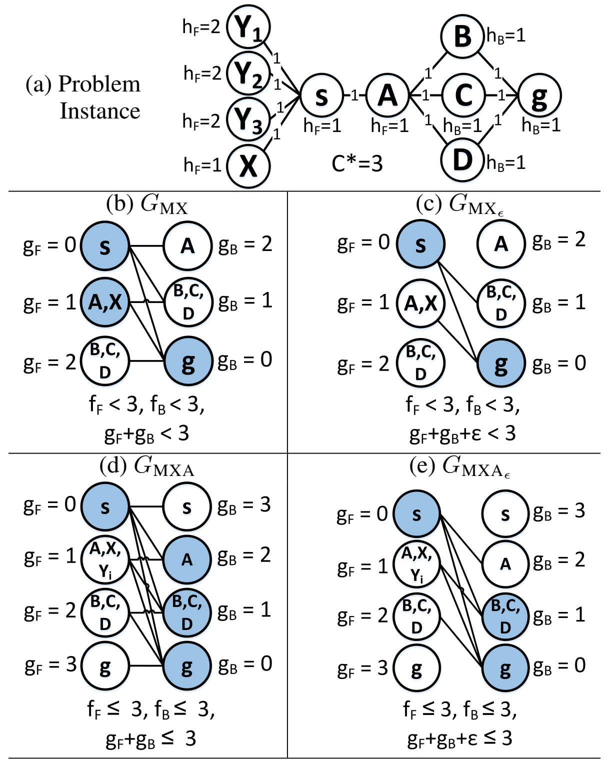

Figure 1: Case Study: Different versions of G_{MX}

e.g., 6 edges connect the cluster with $g_F = 1$ to the one with $g_B = 1$. Figure 1(c) shows G_{MX_ϵ} ($\epsilon = 1$). Due to the addition of ϵ, some edges that exist in G_{MX} no longer exist in G_{MX_ϵ}. For example, the left cluster (vertex) with $g_F = 1$ is connected to all right clusters with $g_B \leq 1$ in G_{MX} but is only connected to the right cluster with $g_B = 0$ in G_{MX_ϵ}.

2.3 The Minimum Vertex-Cover of G_{MX}

Shaham et al. (2017) introduced CalculateMVC() (see their Section 6.5), an algorithm for finding an MVC of a G_{MX}. This algorithm relies on the fact that all such MVCs are *contiguous* and *restrained* in both directions. That is, there exist thresholds $t_F, t_B \in \mathbb{R}$ such that $t_F + t_B = C^*$ ($t_F + t_B + \epsilon = C^*$ for ϵ-case) for which a vertex u in direction D is in the MVC if and only if $g_D(u) < t_D$.

CalculateMVC() iterates over all relevant pairs of values for which $t_F + t_B = C^*$ and finds the pair which induces the MVC. For example, in Figure 1(b) the MVC (colored blue) is induced by $\langle t_F, t_B\rangle = \langle 2, 1\rangle$ and includes only four nodes (s, A, X, g). CalculateMVC() runs in time linear in number of clusters ($O(C^*)$) but assumes that G_{MX} and C^* are given as input. Thus, it can only run post-priori, after C^* was found and the entire G_{MX} was fully built (e.g., by running A* from both sides). Such information is not available to any Bi-HS algorithm during execution. Therefore, Bi-HS algorithms cannot guarantee that the VC they find is minimal. Hence, a main challenge of Bi-HS is to approximate an MVC by using only information available during the search.

3 Finding All Optimal Solutions

A common practice in the heuristic search literature is to halt the search once the *first* optimal solution is found and verified. This problem comprises two tasks: (1) finding a solution of cost C^* and (2) verifying that there are no solutions with cost $< C^*$. Most search algorithms interleave these tasks, completing them in an arbitrary order. The G_{MX} analysis above only handles the second task. Therefore, an MVC of G_{MX} may not capture the extra work needed to complete the first task of finding a solution (but $|\text{MVC}|$ is still a lower bound on the entire search). This is similar to only counting nodes with $f < C^*$ as necessary expansions in unidirectional search, and omitting nodes with $f = C^*$ that are expanded to find the goal (Dechter and Pearl 1985).

In many cases, *all optimal solutions* are required. For example, if not all the problem constraints can be encoded due to privacy issues, competing objectives, partial knowledge, etc. then an external decision maker is needed to choose a solution from the set of all optimal solutions (Byers and Waterman 1984; Arthur et al. 1997; Mahadevan and Schilling 2003). In other cases a solution may become invalid and an alternative solution needs to be obtained quickly (Siegmund, Ng, and Deb 2012; Isermann 1977). We denote these problem spaces by ϵ-ALL-case when knowledge of ϵ exists, and base-ALL-case otherwise.

Finding *all* optimal solutions only consists of a single compound task: verifying that there are no undiscovered solutions with cost $\leq C^*$ (as this includes the task of finding solutions with cost C^*). Thus, we can generalize the analysis in Section 2.1 to the case of finding *all* solutions in a way that allows us to bound the number of expansions required for the *entire* search. In addition, we show below that using this formalization also helps in finding a *first* solution faster.

3.1 G_{MX} for Finding All Optimal Solutions

The first step in generalizing the analysis for the task of finding all solutions is to re-define MEP*s* to use $\leq$ instead of $<$ in the three conditions. Let u and v be nodes in the forward and backward frontiers, respectively. There can be an optimal path (of cost C^*) that goes from s to u to v to g, if:

1. $f_F(u) \leq C^*$
2. $f_B(v) \leq C^*$
3. $g_F(u) + g_B(v) \leq C^*$

Likewise, $g_F(u) + g_B(v) + \epsilon \leq C^*$ is used in the ϵ-ALL-case. We define a pair of states (u, v) to be an MEP for the *all* cases (we call such pairs *must-expand-all* pairs, or MEAPs) if $lb(u, v) \leq C^*$, where $lb(u, v)$ is again the maximum of the three terms.

Theorem 1. *Let $I = \langle G(V, E), s, g\rangle$. A Bi-HS algorithm B will find all optimal paths in I if and only if B expands at least one state from every* MEAP.[1]

proof. **If Case:** Assume that B found all optimal paths but there is an MEAP$\langle u, v\rangle$ where neither u nor v were expanded by B. Consider the two paths: U from s to u with a cost of $g_F(u)$; and V from v to g with the cost of $g_B(v)$. Let $I' = \langle G'(V, E)\rangle, h\rangle$ be a problem instance where $\langle u, v\rangle$ is an edge with cost ϵ. Therefore, there is a path $P = U \cdot V$ from s to g in G'. Since $\langle u, v\rangle$ is an MEAP, the cost of P is $g_F(u) + d(u, v) + g_B(v) = g_F(u) + g_B(v) + \epsilon \leq C^*$. However, $B(I') = B(I) \not\ni P$, contradicting the assumption that all optimal paths from s to g were found by B.

Only-If Case: Assume that B expanded at least one state from every MEAP, and there exists an optimal solution $P = \langle s = p_0, \ldots, p_k = g\rangle$ that was not found. Since the heuristics are admissible, for all $0 \leq i \leq k$, $f_F(p_i) \leq C^*$, $f_B(p_i) \leq C^*$. Since P was not found, there exist nodes $p_i, p_j \in P$, $p_i \neq p_j$, in the forward frontier and backward frontiers of B respectively, when the search terminates. P is an optimal path, thus, $g_F(p_i) + g_B(p_j) + d(p_i, p_j) = C^*$. Since ϵ is a lower bound on the distance between nodes, $g_F(p_i) + g_B(p_j) + \epsilon \leq g_F(p_i) + g_B(p_j) + d(p_i, p_j) = C^*$. Hence $\langle p_i, p_j\rangle$ is an MEAP, contradicting the assumption that B expanded at least one state from every MEAP. □

Note that the proof holds in base-ALL-case if $\epsilon = 0$.

We use the new *must-expand-all* conditions to define two new graphs: G_{MXA} for base-ALL-case, and G_{MXA_ϵ} for ϵ-ALL-case, in a manner similar to G_{MX} and G_{MX_ϵ} respectively, but with the $\leq$ conditions. Importantly, $|\text{MVC}|$ of G_{MXA} and G_{MXA_ϵ} is a lower bound on the number of nodes that must be expanded to complete the joint task of finding *all* optimal solutions and verifying that there are no cheaper solutions. By contrast, $|\text{MVC}|$ of G_{MX} and G_{MX_ϵ} only bounds the minimal number of expansions to complete the (second) task of verifying that no solution with cost $< C^*$ exists.

G_{MXA} and G_{MXA_ϵ} for the example in Figure 1(a) are shown in Figures 1(d) and 1(e), respectively. As can be seen, each vertex has more neighbors due to the use of $\leq$ instead of $<$ in condition 3. For example, the cluster with $gF = 1$ is now also connected to the cluster with $gB = 2$. Furthermore, since conditions 1 and 2 now also have $\leq$, G_{MXA} contains additional clusters (e.g., with $g_F = 0$) and existing clusters may now be composed of additional states (e.g., y_i with $g_F = 1$ are included in G_{MXA} but not in G_{MX}).

Since G_{MXA} includes more edges than G_{MX}, the contiguous partition of their MVCs may be different, as demonstrated in Figure 1. The MVC of G_{MX} (Figure 1(b)) is composed of the vertices $\{s, A, X\}$ in the forward direction and $\{g\}$ in the backward direction. The MVC of G_{MXA} (Figure 1(d)) is composed of vertex s in the forward direction and $\{g, D, C, B, A\}$ in the backward direction. Note that X is part of the MVC of G_{MX} but not a part of the MVC of G_{MXA}.

As a result, existing Bi-HS algorithms that consider G_{MX} when aiming to find a first solution should be modified to consider G_{MXA} when trying to find all optimal solutions. For example, the optimal fraction of fMM(p) for finding all solutions ($\frac{1}{4}$ for Figure 1(a)) is different from the optimal fraction for finding a first solution ($\frac{2}{3}$). Furthermore, in section 4.2 we demonstrate that algorithms which consider G_{MXA} may be even better at finding the first solution.

[1] We assume B is DXBB (See (Eckerle et al. 2017)). We also assume B maintains a frontier of all unexpanded discovered nodes, from which nodes are removed only upon expansion.

Algorithm 1: `LBF` high-level

```
C ← ∞
LB ← min{h_F(s), h_B(g)}
while LB < C do
    C=ExpandLevel(LB,C)
    Increase LB to the next value
return C
```

Algorithm 2: `NBS` Expand Level (LB, C)

```
1  while true do
2      while min f in waiting_D < LB do
3          move best node from waiting_D to ready_D
4      if ready_D ∪ waiting_D empty then
5          Terminate search - no solution was found
6      if ready_F.g + ready_B.g ≤ LB then
7          Expand_D(C) node with min g_D-value in ready_D
8      else
9          if waiting_D.f ≤ LB then
10             move best node from waiting_D to ready_D
11         else
12             return C
```

4 A General Framework Encompassing NBS

Near-Optimal Bidirectional Search (NBS) (Chen et al. 2017) is a robust state-of-the-art non-parametric algorithm that is guaranteed to expand a VC of G_{MX} whose size is at most 2|MVC|. In this section, we introduce a generalization of NBS: a two-level framework which we call the Lower-Bound-Framework (LBF). NBS is a specific implementation of the low level of LBF. We then introduce additional algorithms in this family which differ in their decisions at the low level of LBF.

LBF has two levels. The high level (Algorithm 1) maintains and dynamically increases a global lower bound (LB) on the cost of an optimal solution. It keeps track of all states in the frontiers (OPEN lists) of the two directions of the search. For each node pair $\langle u, v\rangle$, $lb(u, v)$ is defined according to Definition 1 above, depending of course, on the exact case (base-case, ϵ-case etc.). The global lower bound LB is set to be the minimal lb among all pairs.[2] The low level of LBF then needs to select valid nodes for expansion, i.e., nodes that may be part of paths of cost $\leq LB$. All the algorithms in the LBF family discussed in this paper use the same high level, but differ in the low-level selection policy.

4.1 The Low-Level Expansion Policy of NBS

The low-level policy of NBS is based on an approximate VC algorithm (Papadimitriou and Steiglitz 1982) which repeatedly chooses an edge and adds both its endpoints to the VC. Therefore, NBS repeatedly finds a pair $\langle u, v\rangle$ for which $lb(u, v) \leq LB$ and expands *both* u and v. The implementation details of NBS, as done by the original authors (outlined in Algorithm 2) are as follows. The frontier for each direction D is split into two separate queues: *waiting*$_D$ (sorted by f-value), which serves as a gateway to *ready*$_D$ (sorted by g-value). Nodes with a minimal f-value are moved from *waiting*$_D$ to *ready*$_D$, and only nodes from *ready*$_D$ are expanded. In the pseudo codes, every line which includes D is repeated twice, once for each direction. First (Lines 2–3), all nodes for which $f_D(u) < LB$ are moved to *ready*$_D$. Next (Lines 6–7), NBS selects a pair of nodes $u \in$ *ready*$_F$ and $v \in$ *ready*$_B$ for which $g_F(u)+g_B(v) \leq LB$, and expands *both* u and v. If no such pair is found, NBS repeatedly moves a pair of nodes for which $f_F(u) \leq LB$ and $f_F(u) \leq LB$ from *waiting*$_D$ into *ready*$_D$ (Line 10) and continues to look for a pair for which $g_F(u) + g_B(v) \leq LB$. If such a pair is still not found, the low level reports back to the high level that no valid pairs were found, causing LB to be incremented.

Chen et al. (2017) proved three properties of NBS: (1) It is guaranteed to find an optimal solution. (2) It expands at most 2|MVC| states while finding a VC in G_{MX}. (3) No other Bi-HS algorithm can have better worst-case performance.

4.2 Finding All Optimal Solutions with NBS

The original low level used for NBS by Chen et al. (2017) is based on the properties of MEPs which use $< C^*$ in all three conditions. Therefore, NBS first considers nodes with f_F and f_B which are *strictly less* than LB (Line 2). Nodes with f_F and f_B that equal LB are only added *lazily* later (Lines 9–10 of Algorithm 2). We use NBS_F and $\text{NBS}_{F\epsilon}$ (F for *first* solution) to denote the original versions (Algorithm 2) for the base-case and ϵ-case, respectively.

In order to be better suited for for finding *all* solutions we adapt the low-level expansion policy of NBS to be based on MEAPs which have $\leq$ in the three conditions. Specifically, we modify the NBS_F expansion policy to immediately consider all nodes for which $f_D(u) \leq LB$ by changing the $<$ condition in Line 2 of Algorithm 2 to be $\leq$. This change also eliminates Lines 9–11, as such nodes are handled *eagerly* in Line 2. We use NBS_A and $\text{NBS}_{A\epsilon}$ (A for *all* solutions) to denote these new versions which use the modified expansion policy (with $\leq$ in Line 2) and aim to find a vertex cover of G_{MXA} and G_{MXA_ϵ} respectively.

Note that there are many possible ways to implement the low level of NBS in terms of how to move nodes from *waiting*$_D$ to *ready*$_D$. NBS_F and NBS_A are special cases directly inspired by G_{MX} and G_{MXA}.

4.3 Finding a First Solution with NBS_A

An interesting phenomenon is that although NBS_A is designed to find *all* solutions, it may expand fewer nodes than NBS_F, even when finding the first solution. The explanation for this is as follows. The low level of NBS_A utilizes more information about G_{MX} when making a decision. In an iteration where $LB < C^*$, nodes with $f = LB$ are part of G_{MX}, and considering them earlier helps in increasing LB faster, thus finding an MVC faster. In iterations where $LB = C^*$, a

[2]Other Bi-HS algorithms also maintain and increase a global lower bound on the optimal solution, e.g., C in MM and $fLim$ in GBFHS. These bounds use less information than LB of LBF which directly depends on current knowledge on MEP as defined by the G_{MX} theory and therefore is tighter.

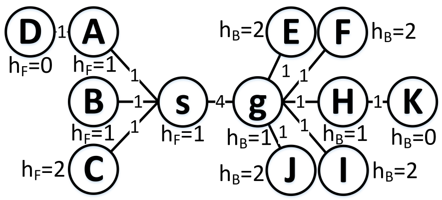

Figure 2: Comparing $\texttt{NBS}_A$ and $\texttt{NBS}_F$

VC of G_{MX} has already been found, and nodes with $f = LB$ can lead to a solution if one was not yet discovered.

An example of this phenomenon is presented in Figure 2, where $C^* = 4$ (the edge between s and g). Both $\texttt{NBS}_F$ and $\texttt{NBS}_A$ begin by expanding $\langle s, g\rangle$ ($LB = 1$), followed by $\langle A, H\rangle$ ($LB = 2$), at which point LB is incremented to 3. For $\texttt{NBS}_F$ *ready*$_B$ will contain only K ($f_F(K) = 2$ while $f = LB = 3$ for all other nodes). $\texttt{NBS}_F$ will expand $\langle B, K\rangle$ before moving other nodes to *ready*$_B$ (using Lines 10–11). Next, it will expand $\langle C, E\rangle$, $\langle D, F\rangle$ and terminate after expanding 10 nodes. By contrast, in $\texttt{NBS}_A$ after setting $LB = 3$ *ready*$_B$ will contain E,F,I,J and K (all with $f \le LB = 3$). Since nodes with lower g-values are expanded first, $\texttt{NBS}_A$ will expand $\langle B, E\rangle$ and $\langle C, F\rangle$, terminating with 8 node expansions, without expanding nodes K and D (since $g_F(D) + g_B(K) = 4 > 3 = LB$). Our experiments below suggest that this phenomenon is rather common in practice.

Note that every pair expanded by $\texttt{NBS}_A$ in every iteration where $LB < C^*$ is an edge of G_{MX}. Thus, $\texttt{NBS}_A$ retains the $2|\texttt{MVC}|$ bound until finding a VC of G_{MX}.

5 Bidirectional Search using Dynamic VC

We now introduce a new family of algorithms called Dynamic Vertex Cover Bidirectional Search (DVCBS). It uses the high level of LBF but conceptually differs from the NBS family in its low-level expansion policy. While NBS always expands both nodes of a chosen MEP, DVCBS works by maintaining a *dynamic* version of G_{MX} (DG_{MX}) and greedily expanding an MVC of the DG_{MX} at each step.

DG_{MX} is defined as follows. Its structure resembles G_{MX}, with two main differences: (1) The full G_{MX} is not available during the search. Instead, DG_{MX} contains only nodes in the forward frontier (generated not expanded) for constructing left vertices, and only nodes from the backward frontier for constructing right vertices. (2) The value of C^* is not known during the search, thus edges of DG_{MX} are defined on pairs $\langle u, v\rangle$ such that $lb(u, v) < LB$. Since $LB \le C^*$, all such pairs are in fact MEPs of G_{MX}.

Note that DG_{MX} shares all the interesting properties of the full G_{MX}. Thus, vertices with the same g-value can be merged to form a weighted vertex (cluster). More importantly, CalculateMVC() can be directly applied to DG_{MX} in time linear in the number of its clusters. This is done in all low-level variants of DVCBS presented next.

5.1 Low-Level Expansion Policy in DVCBS

There are many possible low-level expansion policies based on DG_{MX} and on its MVC. Every node expansion deletes vertices and may add new vertices to DG_{MX}, invalidating the most recently computed MVC. However, computing the MVC every time DG_{MX} changes incurs extra overhead (albeit linear in the number of clusters in DG_{MX}). Thus, an efficient expansion policy should balance between expanding many nodes and maintaining the most up-to-date DG_{MX} and MVC. We experimented with multiple expansion policy variants, and found that an efficient balance between these two extremes is to expand a single cluster (containing all nodes with the same g_F- or g_B-value) in every iteration of the high level. This results in a manageable amount of MVC computations, while working on reasonably up-to-date information. Furthermore, since all vertices in a cluster have the same g-value, LB may increase only after expanding an entire cluster but never before. We only report experimental results for this variant.

DVCBS contains several other decision points. First, there can be several possible MVCs for a given DG_{MX}. Additionally, as mentioned above, one cluster from MVC should be chosen and expanded. Finally, the way we order nodes within the cluster for expansion may affect the number of expansions before reaching a solution when $LB = C^*$. We have experimented with many possible decision choices but report the results in Section 6 using the best variant as follows. Select the cluster with the smallest number of nodes among the clusters with minimal g_F- and g_B-values, among all MVCs. Tie breaking for specific node expansion within a cluster orders nodes according to their order of discovery.

Pseudo code of the low level of DVCBS appears in Algorithm 3. The life cycle of DVCBS includes the following steps: (1) initialize DG_{MX}, (2) CalculateMVC(), (3) choose the cluster of nodes to expand from the MVC, and (4) update DG_{MX}. Steps 2-4 are repeated until either an optimal solution is found or no possible solution exists. To execute efficiently, DVCBS uses data structures denoted as *Cwaiting*$_D$ and *Cready*$_D$, which are similar to the *waiting*$_D$ and *ready*$_D$ queues of NBS, modified to use clusters.

5.2 Variants of DVCBS

Like NBS, DVCBS also has four variants corresponding to the four versions of G_{MX}. The variants that use G_{MX} and G_{MX_ϵ} are denoted by $\texttt{DVCBS}_F$ and $\texttt{DVCBS}_{F\epsilon}$ which *lazily* move nodes with $f_D = LB$ from *Cwaiting*$_D$ to *Cready*$_D$. Likewise, variants that use DG_{MXA} (a dynamic graph based on G_{MXA}, i.e., based on the conditions of MEAPs) can be derived by adapting the low-level expansion policy to G_{MXA} and G_{MXA_ϵ}. Specifically, as was done for NBS, we modify the DVCBS expansion policy to *immediately* consider all nodes for which $f_D(u) \le LB$ by changing the $<$ condition in Line 2 of Algorithm 3 to be $\le$. This change also eliminates Lines 11–13, as we handle such nodes immediately in Line 2. These variants are called $\texttt{DVCBS}_A$ and $\texttt{DVCBS}_{A\epsilon}$.

Here too, $\texttt{DVCBS}_A$ can also be used to find a first solution, sometimes faster than $\texttt{DVCBS}_F$, as we demonstrate using Figure 2. Initially, $LB = 1$. Since no nodes have $f_D < LB$,

Algorithm 3: DVCBS Expand a Level

```
while true do
    while min f in Cwaiting_D < LB do
        Move best cluster from Cwaiting_D to Cready_D
    if Cready_D ∪ Cwaiting_D empty then
        Terminate search - no solution was found
    DG_MX ← BuildDGMX(Cready_D)
    if DG_MX is not empty then
        MVC ← findMVC(DG_MX)
        Choose and Expand a cluster from MVC of
          DG_MX.
    else
        if Cwaiting_D.f ≤ LB then
            Move best cluster from Cwaiting_D to
              Cready_D
        else
            return true
```

$DG_{\text{MX}} = DG_{\text{MXA}} = \{U_F = \{s\}, V_B = \{g\}, E = \{\langle s, g\rangle\}\}$. Assume that both $\texttt{DVCBS}_F$ and $\texttt{DVCBS}_A$ selected s for expansion and so $\{A, B, C\}$ are added to $waiting_F$. Their minimal f-value is 2 (A and B) so $LB = 2$. There are no clusters in $waiting_F$ with $f_F < LB$, thus, {A,B} are moved to $ready_F$ and $DG_{\text{MX}} = DG_{\text{MXA}} = \{U_F = \{A, B\}, V_B = \{g\}, E = \{\langle A, g\rangle, \langle B, g\rangle\}\}$. Therefore, $\{g\}$ is the MVC, and both algorithms expand g and add $\{E, F, H, I, J\}$ to $waiting_B$. Next (LB is still 2), H is added to $ready_B$ and since H is the MVC, it is expanded and K is added to $waiting_B$. Now, $\{K\}$ is the only cluster in $waiting_B$ with $f_B \le LB$. Since $g_B(K) = 2$ and $gmin_F = 1$ ($\{A, B\}$) LB is incremented to 3. At this point the algorithms diverge. DG_{MXA} moves C to $ready_F$ and $\{E, F, I, J, K\}$ to $ready_B$. Thus, DG_{MXA} includes 3 clusters with $f_D \le LB = 3$: $\{A, B, C\}$ with $g_F = 1$ in $ready_F$, and two clusters in $ready_B$: $\{E, J, F, I\}$ with $g_B = 1$, and $\{K\}$ with $g_B = 2$. Thus, $\texttt{DVCBS}_A$ expands cluster $\{A, B, C\}$ (it is the MVC), then, D is generated and expanded and $\texttt{DVCBS}_A$ terminates after expanding a total of 7 nodes (s, g, H, A, B, C and D). By contrast, when $LB = 3$, DG_{MX} contains only two clusters with $f_D < LB = 3$: $\{A, B\}$ (with $g_F = 1$) in $ready_F$ and $\{K\}$ (with $g_F = 2$) in $ready_B$. Thus, $\texttt{DVCBS}_F$ expands K (node C, as well as $\{E, J, F, I\}$ are added to $ready_D$, with $f_D = LB = 3$). Then it expands cluster $\{A, B, C\}$. Next it exapnds D and terminates, after expanding a total of 8 nodes (s, g, H, K, A, B, C and D). Recall that $\texttt{NBS}_F$ expands 10 nodes and $\texttt{NBS}_A$ expands 8 on this example.

5.3 No Upper Bound Guarantees for DVCBS

The most important property of NBS is the 2× bound guarantee. While DVCBS outperforms NBS on average (see experiments below), DVCBS is not bounded in its worst case. A synthetic example and its G_{MX} demonstrate this in Figure 3. The optimal path is $\langle s, X, g\rangle$ of cost $k + (k-1) = 2k-1$. Note that there is a longer path to X via the v_i nodes of cost

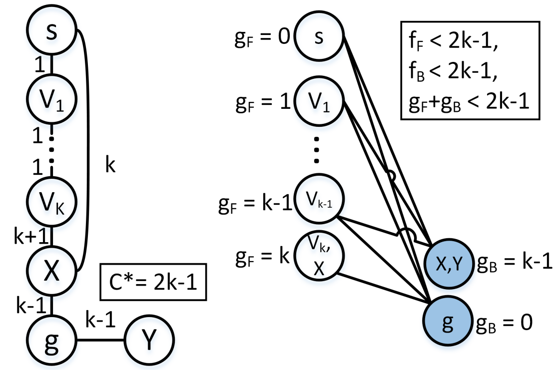

Figure 3: An example for unbounded behavior of DVCBS

$2k+1$. In this example, the MVC of G_{MX} includes three nodes (g, X and Y in the backward direction, all colored blue). We next show that DVCBS never expands Y, and therefore has to expand at least $k+2$ nodes — all connected to Y in G_{MX}. To expand Y, an algorithm needs to generate it by expanding g. If at any point DVCBS chooses to expand g then DG_{MX} will have two nodes in the backward side ($\{X, Y\}$) and a single node in the forward side (s or one of the V_i nodes). Thus, the MVC of DG_{MX} is always in the forward direction (choosing the V_i node), and DVCBS has to expand all of $s, V_1, \ldots, V_{k-1}$ before converging to the size $k+1$ VC of G_{MX}. Otherwise, if g is never chosen for expansion, DVCBS always chooses to expand nodes in the forward direction and it has to expand $k+2$ nodes (s, X and all of the V_is) in order to find a VC. In both cases, DVCBS expands more than k nodes. Since k can be arbitrarily large, DVCBS is not bounded by a constant factor of the MVC.

6 Experimental Evaluation

We ran experiments on four domains: **(1)** 50 **14-Pancake Puzzle** instances with the GAP heuristic (Helmert 2010). To get a range of heuristic strengths, we also used the GAP-n heuristics (for $n = 1 \ldots 3$) where the n smallest pancakes are left out of the heuristic computation. **(2)** The standard 100 instances of the **15 Puzzle** problem (Korf 1985) using the Manhattan Distance heuristic. **(3) Grid**-based pathfinding: 156 maps from Dragon Age Origins (DAO) (Sturtevant 2012), each with different start and goal points (a total of 3150 instances); (4) 50 instances of the 12-disk **4-peg Towers of Hanoi** (TOH4) problem with (10+2), (8+4) and (6+6) additive PDBs (Felner, Korf, and Hanan 2004).

Table 1 presents results averaged over all instances for a representative set of the heuristics we used. The same trends were observed for other heuristics. The left side of the table is for the base-case while the right side is for the ϵ-case. Four low-level expansion policies were executed until all optimal solutions were found: $\texttt{NBS}_F$, $\texttt{NBS}_A$, $\texttt{DVCBS}_F$ and $\texttt{DVCBS}_A$. For comparison reasons we also added A* as a baseline. We report the number of nodes expanded at three different points of the execution, each in a different column, as follows. **(1)** The VC column presents

Domain	Heuristic	Algorithm	**base-case**			**ϵ-case**		
			VC: G_{MX}	first	all: G_{MXA}	VC: G_{MX_ϵ}	first	all: G_{MXA_ϵ}
14 Pancake	GAP	A*	32 (1.22)	**57**	941 (1.17)	32 (1.23)	**57**	941 (1.24)
		NBS$_F$	49 (1.88)	163	1,338 (1.67)	47 (1.83)	147	1,224 (1.61)
		NBS$_A$	44 (1.70)	258	1,106 (1.38)	41 (1.57)	310	932 (1.23)
		DVCBS$_F$	**31 (1.18)**	**106**	**880 (1.10)**	**30 (1.14)**	**121**	832 (1.09)
		DVCBS$_A$	32 (1.24)	191	901 (1.12)	31 (1.18)	284	**793 (1.04)**
	GAP-1	A*	6,410 (1.39)	6,412	81,705 (1.56)	6,404 (1.73)	6,416	81,694 (2.11)
		NBS$_F$	7,184 (1.55)	7,226	80,192 (1.53)	5,870 (1.59)	5,915	62,374 (1.61)
		NBS$_A$	5,656 (1.22)	5,705	61,699 (1.18)	4,332 (1.17)	4,527	45,746 (1.18)
		DVCBS$_F$	5,319 (1.15)	5,341	61,278 (1.17)	4,321 (1.17)	**4,344**	45,206 (1.17)
		DVCBS$_A$	**4,818 (1.04)**	**4,886**	**52,747 (1.01)**	**3,750 (1.01)**	9,955	**38,819 (1.00)**
	GAP-2	A*	322,299 (2.65)	322,378	2,659,657 (3.33)	322,099 (4.15)	322,938	2,659,326 (5.61)
		NBS$_F$	208,648 (1.71)	209,723	1,393,062 (1.74)	137,295 (1.77)	137,719	842,947 (1.78)
		NBS$_A$	151,616 (1.24)	152,046	991,354 (1.24)	96,774 (1.25)	99,773	614,320 (1.30)
		DVCBS$_F$	141,111 (1.16)	141,669	864,611 (1.08)	86,292 (1.11)	**87,012**	493,288 (1.04)
		DVCBS$_A$	**122,054 (1.00)**	**122,587**	**800,105 (1.00)**	**77,595 (1.00)**	168,176	**474,315 (1.00)**
15 Puzzle	MD	NBS$_F$	13,542,536 (N/A)	13,587,955	28,117,879 (N/A)	12,709,517 (N/A)	12,748,107	26,162,236 (N/A)
		NBS$_A$	12,696,359 (N/A)	12,817,989	24,649,233 (N/A)	11,739,393 (N/A)	12,556,299	22,648,690 (N/A)
		DVCBS$_F$	11,863,100 (N/A)	11,940,791	25,717,691 (N/A)	11,589,837 (N/A)	**11,669,720**	24,088,398 (N/A)
		DVCBS$_A$	**11,253,941 (N/A)**	**11,449,406**	**23,276,239 (N/A)**	**10,659,744 (N/A)**	11,933,791	**21,619,261 (N/A)**
Grids DAO	Octile	A*	5,322 (1.25)	**5,406**	5,758 (1.20)	5,322 (1.25)	**5,406**	5,758 (1.20)
		NBS$_F$	6,569 (1.54)	6,686	6,952 (1.45)	6,561 (1.54)	6,677	6,942 (1.44)
		NBS$_A$	6,555 (1.54)	6,888	6,932 (1.44)	6,547 (1.53)	6,880	6,919 (1.44)
		DVCBS$_F$	5,158 (1.21)	**5,546**	5,594 (1.16)	5,158 (1.21)	**5,545**	5,593 (1.16)
		DVCBS$_A$	**5,154 (1.21)**	5,547	**5,590 (1.16)**	**5,152 (1.21)**	5,546	**5,586 (1.16)**
TOH4	10+2	A*	276,081 (2.25)	276,089	353,130 (2.28)	276,081 (2.25)	276,089	353,130 (2.28)
		NBS$_F$	234,165 (1.91)	234,165	291,195 (1.88)	232,509 (1.90)	232,509	288,177 (1.86)
		NBS$_A$	232,268 (1.89)	232,268	288,583 (1.86)	230,108 (1.88)	230,108	285,073 (1.84)
		DVCBS$_F$	225,910 (1.84)	225,910	**273,210 (1.76)**	224,233 (1.83)	224,249	**270,715 (1.74)**
		DVCBS$_A$	**218,820 (1.78)**	**218,820**	280,800 (1.81)	**217,247 (1.77)**	**219,022**	278,286 (1.79)
	6+6	A*	3,239,287 (4.75)	3,268,093	3,674,518 (4.89)	3,239,287 (5.19)	3,268,093	3,674,518 (5.34)
		NBS$_F$	731,446 (1.07)	731,522	796,289 (1.06)	663,136 (1.06)	681,995	732,638 (1.07)
		NBS$_A$	730,562 (1.07)	730,597	795,564 (1.06)	662,424 (1.06)	681,989	732,303 (1.06)
		DVCBS$_F$	704,213 (1.03)	707,679	766,722 (1.02)	636,375 (1.02)	664,469	695,950 (1.01)
		DVCBS$_A$	**690,389 (1.01)**	**691,159**	**757,484 (1.01)**	**627,983 (1.01)**	**660,555**	**690,348 (1.00)**

Table 1: Experimental results of average node expansions across domains

the number of nodes expanded until the algorithm reached a VC of the corresponding G_{MX}. The number reported in parenthesis is the *ratio* (i.e., the relative size) of the discovered VC compared to an oracle (Shaham et al. 2017), that built the entire G_{MX} (by running A* in both directions) and found its exact MVC. Numbers close to 1 indicate nearly optimal VCs. Due to memory limits, some MVCs could not be computed (N/A). **(2)** The *first* column shows the number of nodes expanded until the first solution was found and verified. **(3)** The *all* column gives the number of nodes expanded until *all* optimal solutions were found (i.e., exactly when a VC of $G_{\text{MXA}}/G_{\text{MXA}_\epsilon}$ is found). Here, the ratio relative to the optimal MVC of $G_{\text{MXA}}/G_{\text{MXA}_\epsilon}$ is reported.

Runtime results are reported in Table 2. The node expansion rates of all variants were similar, with very low variance. Therefore, we use the number of node expansions as the measure in the following analysis of the results.

Previous research (Chen et al. 2017; Sturtevant and Felner 2018) reported that NBS tends to outperform and is more robust than A* and other related Bi-HS algorithms (e.g., MM). Table 1 confirms that A* is not as robust as the LBF family. In some cases, e.g., the 15 puzzle, A* failed to solve all instances because memory was exhausted. Except for cases where the heuristic is very good (where MVC might be unidirectional), A*'s performance is much worse than the LBF family in all three measures. See (Shaham et al. 2017) for a deeper study on the relation between A* and MVC.

Since NBS has a 2x bound guarantee, any other algorithm will expand no fewer than half the nodes of NBS, leaving little leeway. Yet, our new algorithms managed to improve upon NBS and the following trends are evident. First, within the NBS family, NBS$_A$ and NBS$_{A\epsilon}$ outperform NBS$_F$ and NBS$_{F\epsilon}$, respectively, in terms of finding a VC of G_{MX} and of G_{MXA}. Moreover, they found the first solution faster than NBS$_F$/NBS$_{F\epsilon}$ in all cases except GAP and DAO.

Second, both DVCBS variants always outperformed the NBS variants in all three measures in the base-case, with DVCBS$_A$ almost always being best. In the ϵ-case, DVCBS$_F$ outperformed NBS$_F$ in all three measures, while DVCBS$_A$ outperformed NBS$_A$ in *VC* and *all*. We note that the VCs discovered by the DVCBS variants were often much closer (e.g., GAP-1; 55% vs. 4%, a factor of 14) to being optimal compared to the VCs discovered by the NBS variants. In fact, in some cases, with a weak heuristic, DVCBS$_A$ managed to find the exact MVC(!) of G_{MX} (a ratio of 1).

Finally, an interesting anomaly occurs with DVCBS$_{A\epsilon}$. It was the fastest to reach a VC of G_{MX_ϵ} but was rarely the fastest to find a first solution; in such cases DVCBS was best among all algorithms. For example, for GAP-2, DVCBS$_{A\epsilon}$ expanded $77,595$ nodes to find a VC of G_{MX_ϵ} while DVCBS found a VC after $86,292$ expansions. However, DVCBS$_{A\epsilon}$ expanded $90,581$ more nodes (totaling $168,176$) before discovering a first solution, while DVCBS$_{F\epsilon}$ expanded only 720 additional nodes (totaling $87,012$). We conjecture that the

Alg	14 Pancake	15 Puzzle	Grids DAO	TOH4
A*	92,697	N/A	**1,821,205**	380,325
$\texttt{NBS}_F$	93,176	250,518	1,567,131	402,616
$\texttt{NBS}_A$	**98,868**	233,166	1,604,500	408,415
$\texttt{DVCBS}_F$	98,448	235,621	1,417,141	418,944
$\texttt{DVCBS}_A$	86,339	**259,756**	1,457,497	**460,368**

Table 2: Average node expansions per second

Domain	BS*	$\texttt{MM}\epsilon$	$\texttt{DVCBS}_{F\epsilon}$	A*
GAP-0	183	149	121	**57**
GAP-1	5,262	5,048	**4,344**	6,416
GAP-2	266,442	119,310	**87,012**	322,938
10+2	**174,936**	303,189	224,249	276,089
6+6	1,599,018	1,120,392	**664,469**	3,268,093
MD	12,001,024	13,162,312	**11,669,720**	N/A
Octile	6,200	7,396	5,545	**5,406**

Table 3: Average expansions for first solution (ϵ-case)

reason is that in the ϵ-case, the frontiers may not be connected (i.e., same node in both frontiers) when a VC is found, and $\texttt{DVCBS}_{A\epsilon}$ must perform many additional node expansions before connecting the frontiers and finding a solution. However, other algorithms seem to perform more expansions before finding a VC, but they are able to connect the frontiers during this process. We intend to study this behavior further in future work.

To summarize, $\texttt{DVCBS}_A$ is clearly the algorithm of choice (among all 4) when all optimal solutions are needed. When only a first solution is needed, $\texttt{DVCBS}_A$ is the best in the base-case, while $\texttt{DVCBS}_{F\epsilon}$ is the best in ϵ-case. Both always outperform any of the NBS variants, despite not having any theoretical guarantees.

We have also compared $\texttt{DVCBS}_{F\epsilon}$ (which is our best variant for finding a first solution in the ϵ-case) to A* as well as to $\texttt{MM}\epsilon$ (Holte et al. 2017) and BS* (Kwa 1989) which are benchmark Bi-HS algorithms. Table 3 presents the average number of node expansions for finding a first solution in the ϵ-case. As can be seen, $\texttt{DVCBS}_{F\epsilon}$ tends to outperform all others, and is certainly the most robust to weaker heuristic.

7 Conclusions and Future Research

We have enriched the family of non-parametric Bi-HS algorithms as well as the family of G_{MX} graphs while also focusing on the problem of finding *all optimal solutions*. We have shown that our new algorithms outperform existing ones. We aim to look deeper in these directions in the future, and study additional variants and their relative performance.

Acknowledgements

This work was supported by Israel Science Foundation (ISF) grant #844/17 to Ariel Felner and Eyal Shimony, by BSF grant #2017692, by NSF grant #1815660 and by the Frankel center for CS at BGU.

References

Arthur, J. L.; Hachey, M.; Sahr, K.; Huso, M.; and Kiester, A. 1997. Finding all optimal solutions to the reserve site selection problem: formulation and computational analysis. *Environmental and Ecological Statistics* 4(2):153–165.

Barley, M. W.; Riddle, P. J.; Linares López, C.; Dobson, S.; and Pohl, I. 2018. GBFHS: A generalized breadth-first heuristic search algorithm. In *SoCS*, 28–36.

Byers, T. H., and Waterman, M. S. 1984. Determining all optimal and near-optimal solutions when solving shortest path problems by dynamic programming. *Operations Research* 32(6):1381–1384.

Chen, J.; Holte, R. C.; Zilles, S.; and Sturtevant, N. R. 2017. Front-to-end bidirectional heuristic search with near-optimal node expansions. In *Proceedings of IJCAI*.

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

Eckerle, J.; Chen, J.; Sturtevant, N. R.; Zilles, S.; and Holte, R. C. 2017. Sufficient conditions for node expansion in bidirectional heuristic search. In *ICAPS*.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *J. Artif. Intell. Res.* 22:279–318.

Helmert, M. 2010. Landmark heuristics for the pancake problem. In *SoCS*.

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

Isermann, H. 1977. The enumeration of the set of all efficient solutions for a linear multiple objective program. *Journal of the Operational Research Society* 28(3):711–725.

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

Kwa, J. B. H. 1989. BS*: An admissible bidirectional staged heuristic search algorithm. *Artif. Intell.* 38(1):95–109.

Mahadevan, R., and Schilling, C. 2003. The effects of alternate optimal solutions in constraint-based genome-scale metabolic models. *Metabolic engineering* 5(4):264–276.

Papadimitriou, C. H., and Steiglitz, K. 1982. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc.

Shaham, E.; Felner, A.; Chen, J.; and Sturtevant, N. R. 2017. The minimal set of states that must be expanded in a front-to-end bidirectional search. In *SoCS*, 82–90.

Shaham, E.; Felner, A.; Sturtevant, N. R.; and Rosenschein, J. S. 2018. Minimizing node expansions in bidirectional search with consistent heuristics. In *SOCS*, 81–98.

Siegmund, F.; Ng, A. H. C.; and Deb, K. 2012. Finding a preferred diverse set of pareto-optimal solutions for a limited number of function calls. In *IEEE World Congress on Computational Intelligence (CEC)*, 2417–2424.

Sturtevant, N. R., and Felner, A. 2018. A brief history and recent achievements in bidirectional search. In *AAAI*.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE Trans. Comput. Intellig. and AI in Games* 4(2):144–148.