Artificial Intelligence 291 (2021) 103405

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Dynamically improved bounds bidirectional search

E.C. Sewell [a], S.H. Jacobson [b,*]

[a] Department of Mathematics and Statistics, Southern Illinois University Edwardsville, Edwardsville, IL 62026-1653, United States of America
[b] Department of Computer Science, 201 N. Goodwin Avenue (MC258), University of Illinois, Urbana, IL 61801, United States of America

ARTICLE INFO

Article history:
Received 23 January 2019
Received in revised form 24 August 2020
Accepted 27 October 2020
Available online 9 November 2020

Keywords:
Heuristic search
Bidirectional search

ABSTRACT

This paper presents a bidirectional search algorithm that dynamically improves the bounds during its execution. It has the property that it always terminates on or before the forward search meets the backward search. Computational experiments on the pancake problem, the sliding tile puzzle, and the topspin problem demonstrate that it is capable of solving problems using significantly fewer node expansions than A* or state-of-the-art bidirectional algorithms such as MM_ϵ and GBFHS.

1. Introduction

When bidirectional heuristic search (BHS) was first introduced by Pohl [15] in the early 1970s, there was optimism that it could significantly outperform unidirectional search. While there have been a number of success stories for BHS, overall the computational results have been disappointing in comparison with unidirectional search, in particular, in comparison with unidirectional A*. This paper presents a BHS algorithm, Dynamically Improved Bounds Bidirectional Search (DIBBS), that improves upon previous BHS algorithms.

Pohl's algorithm, Bidirectional Heuristic Path Algorithm (BHPA) [15], essentially performs two interleaved A* searches, one in the forward direction, which we denote as FA*, and one in the backward direction, which we denote as BA*. One shortcoming of BHPA is that some nodes may be expanded twice, once in each direction. Kaindl and Kainz [12] proved that because of this shortcoming, BHPA may need to expand nearly all of the nodes that are expanded by FA* and by BA*. In such cases, BHPA is inferior to A*. They also proved that if all the f-values (defined in the next section) are distinct, then BHPA must expand at least as many nodes as either FA* or BA* expands. We prove, theoretically and demonstrate computationally, that DIBBS is capable of breaking through this theoretical barrier. Holte et al. [10,11,19] recently introduced a family of "meet in the middle" algorithms which also are capable of breaking through this theoretical barrier, but their approach is quite different than the approach taken here.

In 1989, Kwa [14] presented BS*, which improved upon BHPA by eliminating the necessity of expanding any nodes in both directions. While this was an important theoretical advance, computational results continued to be disappointing. We prove that DIBBS, like BS*, never expands a node in both directions. In fact, DIBBS goes one step better — it terminates before the first node is eligible to be expanded in both directions. Loosely speaking, DIBBS terminates on or before the two searches meet (defined to mean that a node that has been expanded in one direction is eligible to be expanded in the opposite direction). The "meet in the middle" algorithms by Holte et al. [11] mentioned above, when using a consistent heuristic, also have this property, but again their approach is quite different than the approach taken here.

* Corresponding author.
E-mail address: shj@illinois.edu (S.H. Jacobson).

https://doi.org/10.1016/j.artint.2020.103405
0004-3702/

In the same paper where Kaindl and Kainz [12] demonstrated some of the shortcomings of BHPA, they also laid the foundation for improving BHS algorithms. They developed a BHS algorithm that dynamically improves the bounds provided by the heuristics during the search. In 2004, Auer and Kaindl [1] built two BHS algorithms based on this foundation. The first, Max-BS*, uses the improved bounds as the priority function in one of the search directions, but not the other. It does not use the improved bounds for pruning. The second, BiMax-BS$^*_F$, uses the improved bounds for pruning in both directions, but does not use them in the priority function. Our work further builds on the foundation laid by Kaindl and Kainz. First, we show how to refine and improve the dynamic bounds. Second, we show how to seamlessly build the dynamic bound improvement method into a new priority function that permits DIBBS to fully utilize the improved bounds for pruning in both directions and in the priority function.

More recently, Barker and Korf [2] investigated the limitations of front-to-end BHS algorithms. They arrived at two main conclusions. First, "Adding a weak heuristic to a bidirectional bruteforce search cannot prevent it from expanding additional nodes." Second, "With a strong heuristic, a bidirectional heuristic search expands more nodes than a unidirectional heuristic search." In addition, they conjectured that the possible contribution of Kaindl and Kainz's BHS algorithm was to avoid generating nodes with f equal to the cost of an optimal path. We demonstrate computationally that DIBBS can overcome their second conclusion and their conjecture.

One might guess that Barker and Korf's pessimistic pronouncements might have stifled further research on BHS algorithms, but it actually had the opposite effect. In their paper, they postulated the existence of a balanced BHS algorithm that never expands nodes deeper than the solution midpoint, even though such an algorithm did not exist at the time. Holte et al. [10] filled the void by developing the MM algorithm that is guaranteed to meet in the middle. Sharon et al. [19] modified MM to create MM$_\epsilon$, which exploits knowledge about the smallest cost of an edge. A detailed analysis of both algorithms is presented in Holte et al. [11]. Computational results presented in these papers established that it is possible for BHS algorithms to expand fewer nodes than either bidirectional bruteforce search or unidirectional heuristic search. See [21] for a recent survey of bidirectional heuristic search.

MM and MM$_\epsilon$ spawned a flurry of research, resulting in several new BHS algorithms and a deeper understanding of the minimum number of nodes that must be expanded by a BHS algorithm. Eckerle et al. [7] began investigating the minimum number of nodes that must be expanded. They extended the assumptions from undirectional heuristic search to define Deterministic, Expansion-based, Black Box (DXBB) bidirectional algorithms. They characterized pairs of nodes that must be expanded by any DXBB BHS algorithm. Chen et al. [5] continued this vein of research by proving that the minimum number of node expansions for a DXBB BHS algorithm can be determined by finding a Minimum Vertex Cover (MVC) in an auxiliary bipartite graph called G_{MX}. They also developed a BHS algorithm, named Near-Optimal Bidirectional Search (NBS), that makes use of this knowledge. They proved that NBS will never expand more than twice the number of nodes as the size of a MVC of G_{MX} and that no DXBB BHS algorithm can provide a stronger guarantee than this. Shaham et al. [17] constructed an efficient algorithm for finding an MVC of G_{MX}. They also created Fractional MM (fMM), which allows flexibility in choosing the meeting point of the forward and backward searches. Shaham et al. [18] extend the theory regarding the minimum number of node expansions to the case where the cost of the smallest edge is known and constructed the Meet at the Threshold (MT) algorithm that controls the meeting point of the forward and backward search via a threshold parameter. Shperberg et al. [20] created the Dynamic Vertex Cover Bidirectional Search (DVCBS) algorithm, which, similar to NBS, uses information about G_{MX} to guide the search, but unlike NBS, does not have a theoretical guarantee about its worst case performance. Barley et al. [3] created an algorithm named Generalized Breadth-First Heuristic Search (GBFHS), which has great flexibility in controlling where the forward and backward searches meet.

Although the DXBB assumptions do not explicitly state that h_f is only used in the forward direction and h_b is only used in the backward direction, the analysis in [7] implicitly assumes this. Furthermore, DIBBS assumes that the heuristics are consistent, whereas DXBB algorithms do not. DIBBS does not satisfy these assumptions, so it is not subject to the theoretical minimum number of node expansions provided by the MVC of G_{MX}. In fact, in Sections 5 and 6 it is shown that DIBBS often is capable of solving problems while expanding fewer nodes than in a MVC of G_{MX}. It should be emphasized that while DIBBS shares information between the forward and backward searches, it is a front-to-end algorithm, which means that the heuristics are only evaluated from a node to either the start node or the goal node. This is in contrast with front-to-front algorithms, where the heuristics are evaluated from a node to every other node in the frontier of the opposite search direction. Front-to-front evaluations dynamically improve the bounds, but at a very high computational cost. DIBBS dynamically improves the bounds with very little additional computational cost. This paper only addresses front-to-end BHS algorithms.

After this paper was accepted for publication, the authors were made aware of a paper by Sadhukhan [16], in which a similar algorithm was presented. This paper provides more mathematically rigorous proofs of the correctness of the algorithm, a proof that no node is expanded twice, a more careful analysis of which nodes will not be expanded, and more computational testing, including a computational investigation of the minimum number of nodes that must be expanded.

2. Notation and assumptions

Given an undirected graph $G = (V, E)$ with nonnegative edge costs, a start node s, and a goal node t, the problem is to find a lowest cost path from s to t, where the cost of a path is the sum of the cost of the edges on the path. In order

to simplify the presentation and analysis of the algorithm, we assume that there is at least one path from s to t. We now present the definitions that will be used in this paper.

$$
\begin{aligned}
s &= \text{the start node} \\
t &= \text{the goal/target node} \\
P &= \text{a path from } s \text{ to } t \text{ that includes node } v \\
c(u,v) &= \text{the cost of edge } (u,v) \in E \\
d &= \text{the current search direction} = \begin{cases} f & \text{forward search} \\ b & \text{backward search} \end{cases} \\
d' &= \text{the opposite search direction} = \begin{cases} b & \text{if } d = f \\ f & \text{if } d = b \end{cases} \\
g_d^*(v) &= \text{cost of an optimal path from } \begin{cases} s \text{ to } v & \text{if } d = f \\ v \text{ to } t & \text{if } d = b \end{cases} \\
g_d(P,v) &= \text{cost along } P \text{ from } \begin{cases} s \text{ to } v & \text{if } d = f \\ v \text{ to } t & \text{if } d = b \end{cases} \\
h_d(v) &= \text{lower bound on } g_{d'}^*(v) \\
\Delta_d(P,v) &= g_d(P,v) - h_{d'}(v) \\
f_d(P,v) &= g_d(P,v) + h_d(v) \\
f_d^*(v) &= g_d^*(v) + h_d(v) \\
\overline{f}_d(P,v) &= f_d(P,v) + g_d(P,v) - h_{d'}(v) \\
\overline{f}_f^*(v) &= \min\left\{ f_f(P_{sv},v) + g_f(P_{sv},v) - h_b(v) : P_{sv} \text{ is a path from } s \text{ to } v \right\} \\
\overline{f}_b^*(v) &= \min\left\{ f_b(P_{vt},v) + g_b(P_{vt},v) - h_f(v) : P_{vt} \text{ is a path from } v \text{ to } t \right\} \\
C^* &= g_1^*(t) = g_2^*(s) = \text{the length of the shortest path from } s \text{ to } t \\
\text{UB} &= \text{ upper bound on the length of the shortest path from } s \text{ to } t \\
C_d &= \text{the set of closed (expanded) nodes in direction } d \\
O_d &= \text{the set of open (not yet expanded) nodes in direction } d \\
N(v) &= \{w \in V : (v,w) \in E\}
\end{aligned}
$$

The heuristic $h_f(v)$ ($h_b(v)$) is a lower bound on the cost of an optimal path from v to t (s to v). We assume that h_f and h_b are both consistent, meaning

$$
\begin{aligned}
h_f(u) &\le c(u,v) + h_f(v) \quad \forall (u,v) \in E \\
h_b(v) &\le c(u,v) + h_b(u) \quad \forall (u,v) \in E.
\end{aligned}
$$

We now discuss g_f, f_f, $\overline{f}_f$ (g_b, f_b, $\overline{f}_b$ are defined analogously). An asterisk is used on functions to distinguish an optimal value of the function from a possible suboptimal value along some arbitrary path. For example, $g_f(P,v)$ is the cost of path P from the start node s to node v, whereas $g_f^*(v)$ is the cost of the optimal path from s to v. This distinction is necessary for the theoretical analysis of DIBBS. $f_f(P,v)$ is a lower bound on the cost of completing path P to t, i.e., it is a lower bound on the optimal path from s to t that goes through v along path P. The function $\overline{f}_f$ requires more explanation. This definition was motivated by Kaindl and Kainz's [12] work on dynamically improving bounds during a bidirectional search and lies at the heart of the DIBBS algorithm. Let $\Delta_f(P,v) = g_f(P,v) - h_b(v)$ be the difference between the cost of the path P from s to v and the lower bound on the cost of the optimal path from s to v.[1] Thus, $\Delta_f(P,v)$ is the extra cost of the path P compared to the lower bound. Now $\overline{f}_f(P,v)$ can be rewritten as $\overline{f}_f(P,v) = f_f(P,v) + \Delta_f(P,v)$, which is the lower bound on completing P to t plus the extra cost that has already been incurred along P. Note that in and of itself, $\overline{f}_f(P,v)$ does not provide a valid lower bound on completing P to t. However, when $\overline{f}_f$ is used in conjunction with $\overline{f}_b$ within a BHS algorithm, they can be used to prune nodes and to terminate the algorithm, as will be shown in the next two sections. The following example will be used throughout the paper to illustrate the definitions and concepts.

[1] Note: Kaindl and Kainz defined $\text{Diff}_b(v) = g_f(v) - h_b(v)$, i.e., our Δ_f is their Diff_b.

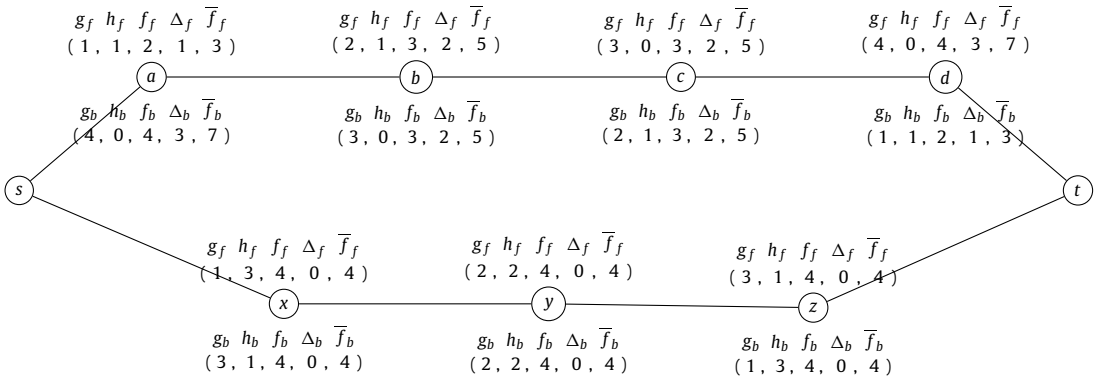

Fig. 1. A graph with unit costs. The optimal path is (s, x, y, z, t) and $C^* = 4$..

Example. Let $G = (V, E)$ be the graph depicted in Fig. 1. Nodes s and t are left unlabeled because some of the values depend on which path is traversed. Let $h_f(s) = 2$, $h_b(s) = 0$, $h_f(t) = 0$, and $h_b(t) = 2$. Let P be the path (s, a, b, c, d, t). For node b, in the forward direction $\Delta_f(P, b) = g_f(P, b) - h_b(b) = 2$ and $\overline{f}_f(P, b) = f_f(P, b) + \Delta_f(P, b) = 3 + 2 = 5$. In the backward direction $\Delta_b(P, b) = g_b(P, b) - h_f(b) = 2$ and $\overline{f}_b(P, b) = f_b(P, b) + \Delta_b(P, b) = 3 + 2 = 5$.

3. Properties of $\overline{f}_d$

This section develops several theoretical properties of $\overline{f}_d$ that will be needed to prove the correctness of DIBBS. The first proposition introduces a useful alternative formula for computing $\overline{f}_d$.

Proposition 1. *Let P be a path from s to t that includes v. Then*

1. $\overline{f}_d(P, v) = 2g_d(P, v) + h_d(v) - h_{d'}(v)$
2. $\overline{f}^*_d(v) = 2g^*_d(v) + h_d(v) - h_{d'}(v)$.

Proof.

$$\begin{aligned}\overline{f}_d(P, v) &= f_d(P, v) + g_d(P, v) - h_{d'}(v)\\ &= g_d(P, v) + h_d(v) + g_d(P, v) - h_{d'}(v)\\ &= 2g_d(P, v) + h_d(v) - h_{d'}(v),\end{aligned}$$

which proves the first part of the proposition. The second part follows from the first part since the term $h_d(v) - h_{d'}(v)$ has no connection with the path P, and hence, the path that minimizes g_d also minimizes $\overline{f}_d$. □

Proposition 2 follows immediately from Proposition 1 and from the definition of $\overline{f}^*_d$. It states that the optimal path from s to v also is a path that minimizes $\overline{f}_f(P, v)$, and vice versa. Because DIBBS uses $\overline{f}_d$ as the priority function for selecting the next node to be expanded, it is crucial to know that when the path that minimizes $\overline{f}_f(P, v)$ is found, then the optimal path to v has also been found. Proposition 2 will be used in Lemma 8 to prove this fact.

Proposition 2. *Let P be a path from s to t that includes v. Then*

$$g_d(P, v) = g^*_d(v) \iff \overline{f}_d(P, v) = \overline{f}^*_d(v).$$

The next proposition is an immediate consequence of Proposition 2 and the definition of $\overline{f}^*_d$.

Proposition 3. $\overline{f}^*_d(v) = f^*_d(v) + g^*_d(v) - h_{d'}(v)$.

Proposition 4 shows that for every node v on a path from s to t, the sum of $\overline{f}_f(P, v)$ and $\overline{f}_b(P, v)$ equals twice the cost of the path. This result will be used in Theorems 11 and 12 to obtain the termination criteria for DIBBS.

Proposition 4. *Let P be a path from s to t that includes v. Then*

$$g_f(P,t) = \frac{\overline{f}_f(P,v) + \overline{f}_b(P,v)}{2}.$$

Proof.

$$\begin{aligned}\overline{f}_f(P,v) + \overline{f}_b(P,v) &= \left[2g_f(P,v) + h_f(v) - h_b(v)\right] + \left[2g_b(P,v) + h_b(v) - h_f(v)\right] \\ &= 2\left(g_f(P,v) + g_b(P,v)\right) \\ &= 2g_f(P,t).\end{aligned}$$

The first equality follows from Proposition 1 and the third equality follows from the definitions of $g_f(P,v)$ and $g_b(P,v)$. □

Example *(cont.).* Let $P = (s,a,b,c,d,t)$ in the graph $G = (V,E)$ depicted in Fig. 1. Then

$$g_f(P,t) = 5 = \frac{\overline{f}_f(P,b) + \overline{f}_b(P,b)}{2} = \frac{5+5}{2}$$

and the same relationship holds true for every other node on P.

Proposition 5 demonstrates that $\overline{f}_b$ is monotonically increasing along any path traversed from s to t. It will be used in Lemma 9 to show that the minimum value of $\overline{f}_f$ among all open nodes increases monotonically during the execution of DIBBS, which in turn will be used in Theorem 10 to prove that when a node has been expanded, the shortest path to that node has been found.

Proposition 5. *Let $P = (v_0 = s, v_1, v_2, \ldots, v_k = t)$ be a path from s to t. Then*

$$\overline{f}_f(P,v_{i+1}) - \overline{f}_f(P,v_i) = \overline{f}_b(P,v_i) - \overline{f}_b(P,v_{i+1}) \geq 0.$$

Hence $\overline{f}_f$ is monotonically increasing as P is traversed from s to t and $\overline{f}_b$ is monotonically increasing as P is traversed from t to s.

Proof. Proposition 4 implies that $\overline{f}_f(P,v_i) + \overline{f}_b(P,v_i) = 2g_f(P,t) = \overline{f}_f(P,v_{i+1}) + \overline{f}_b(P,v_{i+1})$, hence $\overline{f}_f(P,v_{i+1}) - \overline{f}_f(P,v_i) = \overline{f}_b(P,v_i) - \overline{f}_b(P,v_{i+1})$. We now must show that these two terms are nonnegative.

$$\begin{aligned}\overline{f}_f(P,v_{i+1}) - \overline{f}_f(P,v_i) &= \left[f_f(P,v_{i+1}) + g_f(P,v_{i+1}) - h_b(v_{i+1})\right] \\ &\quad - \left[f_f(P,v_i) + g_f(P,v_i) - h_b(v_i)\right] \\ &= \left[g_f(P,v_{i+1}) + h_f(v_{i+1}) + g_f(P,v_{i+1}) - h_b(v_{i+1})\right] \\ &\quad - \left[g_f(P,v_i) + h_f(v_i) + g_f(P,v_i) - h_b(v_i)\right] \\ &= \left[g_f(P,v_{i+1}) - g_f(P,v_i) + h_f(v_{i+1}) - h_f(v_i)\right] \\ &\quad + \left[g_b(P,v_{i+1}) - g_b(P,v_i) + h_b(v_i) - h_b(v_{i+1})\right] \\ &= \left[c(v_i,v_{i+1}) + h_f(v_{i+1}) - h_f(v_i)\right] \\ &\quad + \left[c(v_i,v_{i+1}) + h_b(v_i) - h_b(v_{i+1})\right] \\ &\geq 0.\end{aligned}$$

The first equation expands the definition of $\overline{f}_f$, the second equation expands the definition of f_f, the third equation is obtained by rearranging terms, and the fourth equation substitutes $c(v_i,v_{i+1})$ for $g_f(P,v_{i+1}) - g_f(P,v_i)$. The last inequality follows from the consistency of h_f and h_b. □

Example *(cont.).* Let $P = (s,a,b,c,d,t)$ in the graph $G = (V,E)$ depicted in Fig. 1. Given $h_f(s) = 2$, $h_b(s) = 0$, $h_f(t) = 0$, and $h_b(t) = 2$. Then

$$\begin{aligned}\overline{f}_f(P,s) &= 2g_f(P,s) + h_f(s) - h_b(s) = 0 + 2 - 0 = 2, \\ \overline{f}_f(P,t) &= 2g_f(P,t) + h_f(t) - h_b(t) = (2)(5) + 0 - 2 = 8,\end{aligned}$$

and the sequence of $\overline{f}_f$-values is

$$\left(\overline{f}_f(P,s), \overline{f}_f(P,a), \overline{f}_f(P,b), \overline{f}_f(P,c), \overline{f}_f(P,d), \overline{f}_f(P,t)\right) = (2,3,5,5,7,8),$$

which is monotonically increasing.

The two inequalities in Proposition 6 serve as the basis for Kaindl and Kainz's [12] method of dynamically improving lower bounds during BHS. Auer and Kaindl defined $F_f(v) = \max\left(f_f(v), f_{\min_b} + g_f(v) - h_b(v)\right)$, where $f_{\min_b}$ is the minimum value of f_b among all open nodes in the backward direction ($F_b(v)$ defined similarly). Their Max-BS* algorithm uses F_d as the priority function in one of the search directions. Their BiMax-BS$^*_F$ algorithm uses F_d for pruning in both directions, but does not use it as a priority function.

Proposition 6. *Let P be a path from s to t that includes v and w, such that v precedes w on P. Then*

1. $g_f(P,t) \geq f_f(P,v) + g_b(P,w) - h_f(w) = f_f(P,v) + \Delta_b(P,w)$
2. $g_b(P,t) \geq f_b(P,w) + g_f(P,v) - h_b(v) = f_b(P,w) + \Delta_f(P,v)$.

Proof. We prove the first part of the proposition. The second part can be proven similarly.

$$\begin{aligned}
2g_f(P,t) &= \overline{f}_f(P,v) + \overline{f}_b(P,v) && \text{(by Proposition 4)}\\
&= f_f(P,v) + g_f(P,v) - h_b(v) && \text{(by definition of } \overline{f}_f\text{)}\\
&\quad + f_b(P,v) + g_b(P,v) - h_f(v) && \text{(by definition of } \overline{f}_b\text{)}\\
&= f_f(P,v) + g_f(P,v) - h_b(v)\\
&\quad + g_b(P,v) + h_b(v) + g_b(P,v) - h_f(v) && \text{(by definition of } f_b\text{)}\\
&= f_f(P,v) + \left[g_f(P,v) + g_b(P,v)\right] + g_b(P,v) - h_f(v)\\
&= f_f(P,v) + g_f(P,t) + g_b(P,v) - h_f(v) && \text{(by definition of } g_d(P,v)\text{)}
\end{aligned}$$

Therefore,

$$\begin{aligned}
g_f(P,t) &= f_f(P,v) + g_b(P,v) - h_f(v)\\
&\geq f_f(P,v) + g_b(P,w) - h_f(w).
\end{aligned}$$

The final inequality can be obtained by supposing that v_i and v_{i+1} are two consecutive nodes on the path and then showing that $\left[g_b(P,v_{i+1}) - h_f(v_{i+1})\right] - \left[g_b(P,v_i) - h_f(v_i)\right] = c(v_i, v_{i+1}) + h_f(v_i) - h_f(v_{i+1}) \geq 0$ due to the consistency of h_f. □

Example *(cont.).* For the graph depicted in Fig. 1. Proposition 6 implies that any path P from s to t that includes b and d, in that order, must cost at least $f_f(P,b) + \Delta_b(P,d) = 3 + 1 = 4$.

4. Bidirectional search algorithm

This section presents the bidirectional algorithm and its properties. Capital letters G_d and $\overline{F}_d$ are used in the algorithm instead of g_d and $\overline{f}_d$, respectively, in order to distinguish between values computed by the algorithm and values computed by the original mathematical definition. The *Expand Backward* function, which is analogous to the *Expand Forward* function, is not shown here.

Dynamically Improved Bounds Bidirectional Search (DIBBS) Algorithm

$G_d(v) = \infty, \overline{F}_d(v) = \infty$ for $d = f, b$ and $\forall v \in V$
$G_f(s) = 0, \overline{F}_f(s) = \overline{f}^*_f(s), \overline{F}_{\min_f} = 0, G_b(t) = 0, \overline{F}_b(t) = \overline{f}^*_b(t), \overline{F}_{\min_b} = 0$
$C_f = C_b = \emptyset$ // C_d is the set of closed nodes in direction d.
$O_f = O_b = V$ // O_d is the set of open nodes in direction d.
UB $= \infty$
while UB $> \frac{\overline{F}_{\min_f} + \overline{F}_{\min_b}}{2}$
  Choose a search direction d
  if $d = f$
    Expand Forward
  else
    Expand Backward
  endif
endwhile

Expand Forward

$v = \arg\min\left\{\overline{F}_f(v) : v \in O_f\right\}$ // Choose open node with the smallest $\overline{F}_f$ for expansion.

Table 1
DIBBS applied to the graph depicted in Fig. 1.

Iter.	C_f	O_f	$\overline{F}_{\min_f}$	C_b	O_b	$\overline{F}_{\min_b}$	UB	Selected	
								d	Node
1	$\emptyset$	$(s,2)$	2	$\emptyset$	$(t,2)$	2	∞	f	s
2	s	$(a,3),(x,4)$	3	$\emptyset$	$(t,2)$	2	∞	b	t
3	s	$(a,3),(x,4)$	3	t	$(d,3),(z,4)$	3	∞	f	a
4	s,a	$(b,5),(x,4)$	4	t	$(d,3),(z,4)$	3	∞	b	d
5	s,a	$(b,5),(x,4)$	4	t,d	$(c,5),(z,4)$	4	∞	f	x
6	s,a,x	$(b,5),(y,4)$	4	t,d	$(c,5),(z,4)$	4	∞	f	y
7	s,a,x,y	$(b,5),(z,4)$	4	t,d	$(c,5),(z,4)$	4	4	Terminate	

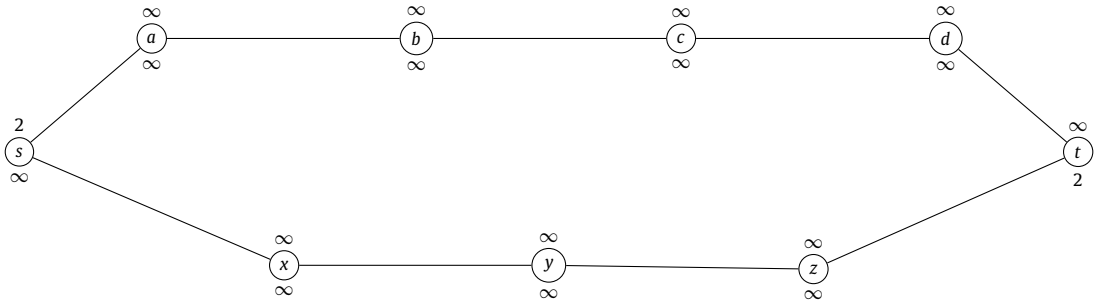

Fig. 2. The graph at the beginning of the first iteration of DIBBS. The label above each node is $\overline{F}_f$. The label below each node is $\overline{F}_b$.

$C_f = C_f \cup \{v\},\ O_f = O_f \setminus \{v\}$ // Move v from the open nodes to the closed nodes.
for $w \in N(v) \setminus C_f$
  if $\overline{F}_f(w) > 2\left(G_f(v) + c(v,w)\right) + h_f(w) - h_b(w)$
    $\overline{F}_f(w) = 2\left(G_f(v) + c(v,w)\right) + h_f(w) - h_b(w)$
    $G_f(w) = G_f(v) + c(v,w)$
    $\text{pred}(w) = v$
    $\text{UB} = \min\left(\text{UB}, G_f(w) + G_b(w)\right)$
  endif
endfor
$\overline{F}_{\min_f} = \min\left\{\overline{F}_f(v) : v \in O_f\right\}$

The overall approach of DIBBS is similar to many other BHS algorithms that have appeared in the literature. It conducts two searches: a forward search starting from s and a backward search starting from t. It maintains a closed and open set of nodes in both directions. Closed nodes are the ones that have already been expanded. At each iteration, a decision is made whether to expand a node in the forward direction or the backward direction. DIBBS is flexible in how that decision is made. One possible criteria will be presented below. Once a direction has been selected, an open node in that direction that has the smallest value of the priority function is chosen to be expanded. DIBBS differs from other BHS algorithms in that it uses $\overline{f}_d$ as the priority function, as opposed to f_d, which is used by many other BHS algorithms. The expansion of a node is done similar to other BHS algorithms, with the exception that $\overline{f}_d$ rather than f_d is updated (because $\overline{f}_d$ is used to order the open lists). Before the next iteration is performed, a termination criteria is checked. DIBBS' termination criteria differs in two respects from many other BHS algorithms. First, some other algorithms do not terminate until one of the open sets is empty. DIBBS can be terminated while both open sets are nonempty. Second, DIBBS uses $\overline{f}_d$ in its termination criterion. One further difference is that DIBBS requires consistent heuristics. If the heuristics are inconsistent, then DIBBS may return a suboptimal path, as demonstrated in an example in Section 6.

Example *(cont.).* Table 1 displays the execution of DIBBS on the graph depicted in Fig. 1. Each line in the table represents the values of the variables at the beginning of an iteration. The entries in the O_f (O_b) column contain the name of the node and the current value of $\overline{F}_f$ ($\overline{F}_b$). The last two columns show which direction, d, was selected and which node was selected to be expanded within that direction. For clarity, nodes with $\overline{F}_d(\cdot) = \infty$ are not included in O_d. If $\overline{F}_{\min_f} \leq \overline{F}_{\min_b}$, then the forward direction was selected, otherwise the backward direction was selected. The execution is also illustrated in Figs. 2, 3, 4, and 5.

DIBBS requires several data structures to be efficiently implemented. In particular, the data structures must be able to choose the open node with the minimum $\overline{F}_d$-value, must maintain a list of both open and closed nodes, and must be able

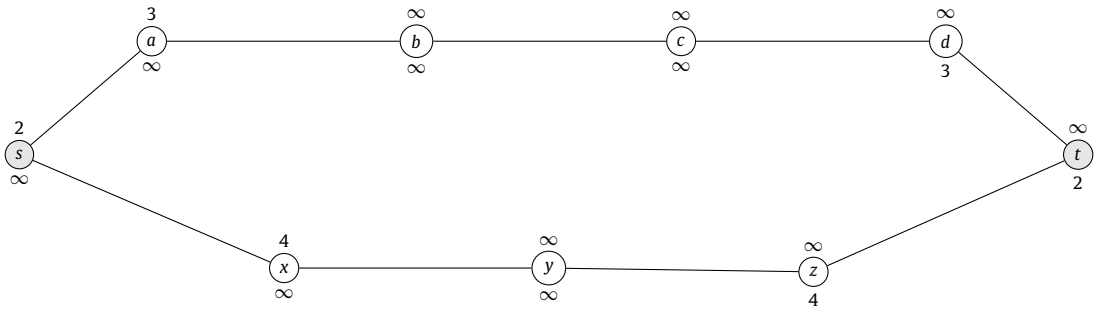

Fig. 3. The graph at the beginning of the third iteration of DIBBS. The closed nodes are shaded.

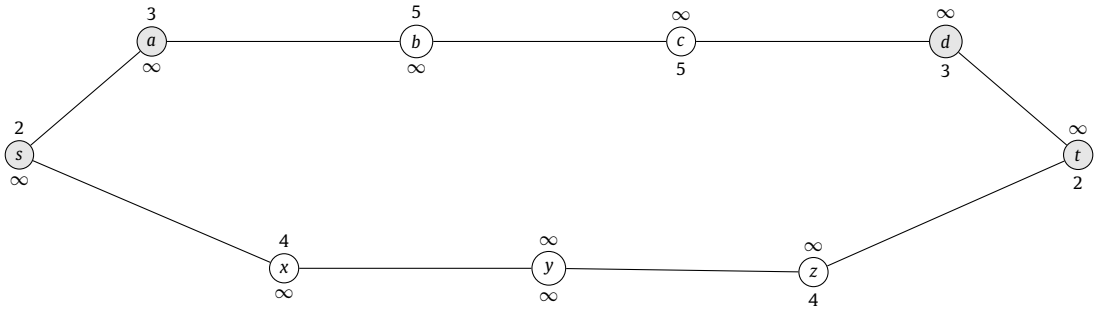

Fig. 4. The graph at the beginning of the fifth iteration of DIBBS. The closed nodes are shaded.

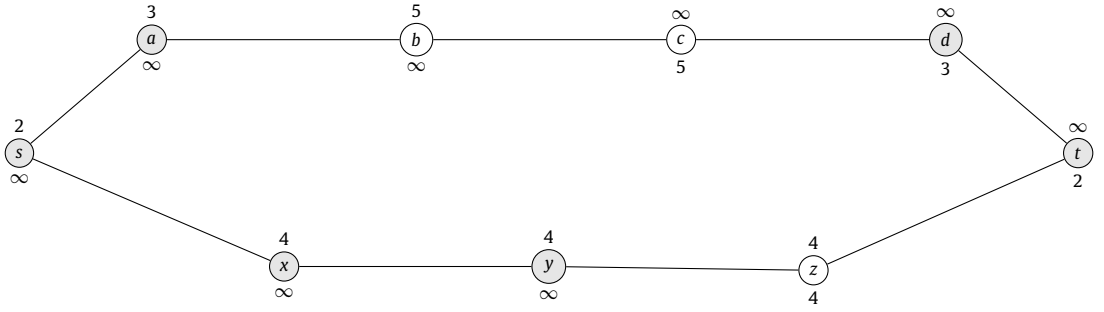

Fig. 5. The graph at the beginning of the seventh iteration of DIBBS. The algorithm terminates because a path of length four has been found resulting in the termination criteria being satisfied.

to recognize if a node is in either of those sets. Our implementation uses two heaps, one for the forward direction and one for the backward direction, to choose the next node to be expanded. It stores all the nodes generated in an unordered list and uses a hash table to determine if a given node is in that list.

Sadhukhan [16] developed an algorithm named BAE* that is similar to DIBBS, although he used different notation. He defined $TE_d(v) = 2g_d(v) + h_d(v) - h_{d'}(v) - h_0$, where $h_0 = h_f(s)$, and used TE as the priority function for selecting the next node to be expanded. The definition of $\overline{f}_d$ and TE_d only differ by a constant, namely $-h_0$, so the two methods are equivalent. One other minor difference between the two algorithms is that BAE* switches directions every iteration, i.e., a node is expanded in the forward direction on odd-numbered iterations and in the backwards direction on even-numbered iterations.

We now prove the correctness of DIBBS. Several of the properties are only stated and proved for the forward search. It is to be understood that an analogous property can be stated and proved for the backward search. We begin with the observation that DIBBS correctly computes the functions $g_f(P, v)$ and $\overline{f}_f(P, v)$.

Lemma 7. *In DIBBS, let P_{sv} be the current path from s to v (assuming that such a path has been found), then*

1. $G_f(v) = g_f(P_{sv}, v)$
2. $\overline{F}_f(v) = \overline{f}_f(P_{sv}, v)$.

DIBBS uses $\overline{f}_d$ as the priority function for selecting the next node to be expanded. As such, the forward search explicitly finds a path from s to v that minimizes $\overline{f}_f(P, v)$. Lemma 8 proves that it implicitly and simultaneously finds the optimal path from s to v.

Lemma 8. *In DIBBS,* $G_f(v) = g_f^*(v) \iff \overline{F}_f(v) = \overline{f}_f^*(v)$.

Proof. Let P_{sv} be the current path from s to v found by the algorithm (i.e., as stored in pred$(\cdot)$). Then $G_f(v) = g_f(P_{sv}, v)$ and $\overline{F}_f(v) = \overline{f}_f(P_{sv}, v)$ by Lemma 7. Therefore,

$$\begin{aligned} G_f(v) = g_f^*(v) &\iff g_f(P_{sv}, v) = g_f^*(v) \\ &\iff \overline{f}_f(P_{sv}, v) = \overline{f}_f^*(v) \\ &\iff \overline{F}_f(v) = \overline{f}_f^*(v), \end{aligned}$$

where the second line follows from the first line by Proposition 2. □

Lemma 9 will be used in Theorem 10 to prove that a node will be expanded at most once in a given direction.

Lemma 9. *During the execution of DIBBS,* $\overline{F}_{\min_f}$ *and* $\overline{F}_{\min_b}$ *increase monotonically.*

Proof. Let $\overline{F}^b_{\min_f}$ be the value of $\overline{F}_{\min_f}$ at the beginning of an arbitrary iteration and $\overline{F}^e_{\min_f}$ be the value of $\overline{F}_{\min_f}$ at the end of that iteration. Similarly, let $\overline{F}^b_f(u)$ be the value of $\overline{F}_f(u)$ at the beginning of the iteration and $\overline{F}^e_f(u)$ be the value of $\overline{F}_f(u)$ at the end of the iteration. If the expansion is in the backward direction, there is nothing to prove, because $\overline{F}_{\min_f}$ does not change during the iteration. Therefore, assume that v has been chosen to be expanded in the forward direction and let P_{sv} be the current path from s to v. Thus $\overline{F}^b_f(v) = \overline{F}^b_{\min_f}$. Let $w = \arg\min\left\{\overline{F}^e_f(w) : w \in O_f \setminus \{v\}\right\}$ be an open node with the smallest $\overline{F}_f$ value at the end of the iteration. If $\overline{F}_f(w)$ was not modified during the iteration, then $\overline{F}^b_{\min_f} = \overline{F}^b_f(v) \le \overline{F}^b_f(w) = \overline{F}^e_f(w) = \overline{F}^e_{\min_f}$ by the definition of v. If $\overline{F}_f(w)$ was modified during the iteration, then

$$\begin{aligned} \overline{F}^e_{\min_f} &= \overline{F}^e_f(w) \\ &= 2\left(G_f(v) + c(v, w)\right) + h_f(w) - h_b(w) \\ &= 2\left(g_f(P_{sv}, v) + c(v, w)\right) + h_f(w) - h_b(w) + \left[h_f(v) - h_f(v)\right] + \left[h_b(v) - h_b(v)\right] \\ &= \left[g_f(P_{sv}, v) + h_f(v) + g_f(P_{sv}, v) - h_b(v)\right] \\ &\quad + \left[c(v, w) + h_f(w) - h_f(v)\right] + \left[c(v, w) + h_b(v) - h_b(w)\right] \\ &\ge \overline{f}_f(P_{sv}, v) \\ &= \overline{F}^b_f(v) \\ &= \overline{F}^b_{\min_f}. \end{aligned}$$

The second equality follows from the update operation in the Expand Forward function. In the third equality, $g_f(P_{sv}, v)$ is substituted for $G_f(v)$ by Lemma 7. The fifth inequality is obtained by substituting $\overline{f}_f(P_{sv}, v)$ for $2g_f(P_{sv}, v) + h_f(v) - h_b(v)$ and by the consistency of h_f and h_b. The sixth equality follows from Lemma 7.

The proof in the backward direction follows in a similar manner. □

Example *(cont.).* Examination of the columns labeled $\overline{F}_{\min_f}$ and $\overline{F}_{\min_b}$ in Table 1 illustrates that these two variables increase monotonically when DIBBS is executed on the graph depicted in Fig. 1.

Theorem 10 is the core of the proof that DIBBS works correctly. Parts (1) and (2) state that when a node u is chosen to be expanded in the forward direction (i.e., put into C_f), then the current path to u minimizes $\overline{f}_f(P, u)$, and consequently is optimal. Furthermore, Parts (1) and (2) imply that u will not need to be expanded again in the forward direction. Part (3) is needed for the proofs of (1) and (2). Part (4), which provides information on how UB is updated, will be used in Theorem 12 to prove that DIBBS terminates on or before the forward and backward searches meet.

Theorem 10. *DIBBS maintains the following invariants.*

Inv$_1$: $u \in C_f \implies \overline{F}_f(u) = \overline{f}_f^*(u)$.

Inv$_2$: $u \in C_f \Longrightarrow G_f(u) = g_f^*(u)$.

Inv$_3$: *If $v \in O_f$ and there exists a shortest path from s to v such that all the nodes on the path are in C_f (excluding v), then $G_f(v) = g_f^*(v)$.*

Inv$_4$: *If $N(v) \cap C_f \neq \emptyset$, then $UB \leq G_f(v) + G_b(v)$.*

Proof. The proof proceeds by induction on the iteration. Clearly, the invariants are satisfied prior to the first iteration. Now assume that the invariants are satisfied at the beginning of an arbitrary iteration, say iteration i, i.e., assume that $\text{Inv}_1(i)$, $\text{Inv}_2(i)$, $\text{Inv}_3(i)$, and $\text{Inv}_4(i)$ are all true. Let C_f^i (O_f^i) be the set of closed (open) nodes at the beginning of iteration i. Suppose v is the next node chosen to enter C_f^i, i.e., $v = \arg\min\left\{\overline{F}_f(v) : v \in O_f^i\right\}$. Let $C_f^{i+1} = C_f^i \cup \{v\}$, $O_f^{i+1} = O_f^i \setminus \{v\}$. We need to show that the invariants $\text{Inv}_1(i+1)$, $\text{Inv}_2(i+1)$, $\text{Inv}_3(i+1)$, and $\text{Inv}_4(i+1)$ will be satisfied at the end of the iteration, i.e., the invariants hold for C_f^{i+1} and O_f^{i+1}. The overall strategy of the proof is to use $\text{Inv}_1(i)$, $\text{Inv}_2(i)$, $\text{Inv}_3(i)$ to prove $\text{Inv}_1(i+1)$, after which $\text{Inv}_1(i+1)$ is used to prove $\text{Inv}_2(i+1)$, and then $\text{Inv}_2(i+1)$ is used to prove $\text{Inv}_3(i+1)$.

1. We need to prove that if $u \in C_f^{i+1}$, then $\overline{F}_f(u) = \overline{f}_f^*(u)$. If $u \neq v$, then $u \in C_f^i$. The algorithm does not change $\overline{F}_f(u)$ during this iteration, so $\overline{F}_f(u) = \overline{f}_f^*(u)$ by the induction hypothesis $\text{Inv}_1(i)$. We now need to show that $\overline{F}_f(v) = \overline{f}_f^*(v)$. Let P^* be a path from s to v that minimizes $f_f(P, v) + g_f(P, v) - h_b(v)$.
   Case 1: $P^* \subseteq C_f^{i+1}$. In this case, P^* is a shortest path from s to v such that all nodes on the path are in C_f^i, excluding v. The induction hypothesis $\text{Inv}_3(i)$ implies $G_f(v) = g_f^*(v)$. Then $\overline{F}_f(v) = \overline{f}_f^*(v)$ by Lemma 8.
   Case 2: $P^* \not\subseteq C_f^{i+1}$. Let x be the first node on P^* that is not in C_f^{i+1}. The induction hypothesis $\text{Inv}_3(i)$ implies $G_f(x) = g_f^*(x)$, which in turn implies $\overline{F}_f(x) = \overline{f}_f^*(x)$ by Lemma 8. By supposition, $v = \arg\min\left\{\overline{F}_f(v) : v \in O_f^i\right\}$, hence $\overline{F}_f(v) \leq \overline{F}_f(x) = \overline{f}_f^*(x)$. On the other hand, $\overline{f}_f$ is monotonically nondecreasing as P^* is traversed from s to v, hence $\overline{f}_f^*(x) = \overline{f}_f(P^*, x) \leq \overline{f}_f(P^*, v) = \overline{f}_f^*(v) \leq \overline{F}_f(v)$. Combining these two inequalities results in
   $$\overline{F}_f(v) \leq \overline{F}_f(x) = \overline{f}_f^*(x) = \overline{f}_f\left(P^*, x\right) \leq \overline{f}_f\left(P^*, v\right) = \overline{f}_f^*(v) \leq \overline{F}_f(v).$$
   Consequently, the inequalities actually are equalities, i.e.,
   $$\overline{F}_f(v) = \overline{F}_f(x) = \overline{f}_f^*(x) = \overline{f}_f\left(P^*, x\right) = \overline{f}_f\left(P^*, v\right) = \overline{f}_f^*(v),$$
   which completes the proof of this case.
2. This follows immediately from the invariant $\text{Inv}_1(i+1)$ and Lemma 8.
3. Let $w \in O_f^{i+1}$ and suppose there exists a shortest path P^* from s to w such that all the nodes on P^* are in C_f^{i+1} (excluding w). Let u be the second to last node on P^*. Now $u \in C_f^{i+1}$ implies u has been expanded, which implies $G_f(w) \leq G_f(u) + c(u, w)$. The invariant $\text{Inv}_2(i+1)$ implies that $G_f(u) = g_f^*(u)$, hence
   $$G_f(w) \leq g_f^*(u) + c(u, w) = g_f\left(P^*, w\right) = g_f^*(w).$$
   But $G_f(w)$ can never be less than $g_f^*(w)$, thus $G_f(w) = g_f^*(w)$.
4. Suppose $N(v) \cap C_f^{i+1} \neq \emptyset$. Among all the nodes in $N(v) \cap C_f^{i+1}$, let u be the last one expanded in the forward direction that modified $G_f(v)$. When u was expanded in the forward direction, UB was set to be less than or equal to $G_f(v) + G_b(v)$. By the choice of u, $G_f(v)$ has not been modified since the iteration when u was expanded. If $G_b(v)$ has been modified since u was expanded in the forward direction, it must have been modified in Expand Backward. Let w be the last node that modified $G_b(v)$. When $G_b(v)$ was modified, UB was set to be less than or equal to $G_f(v) + G_b(v)$. □

Theorem 11 implies that if a path from s to t goes through a node that is not in $C_f \cup C_b$, then the cost of that path must be greater than or equal to $\left(\overline{F}_{\min_f} + \overline{F}_{\min_b}\right)/2$. This verifies that DIBBS' termination criteria is correct.

Theorem 11. *Let $v \in O_f \cap O_b$ and P be a shortest path from s to t that includes v. Then*

$$g_f(P, t) \geq \frac{\overline{F}_{\min_f} + \overline{F}_{\min_b}}{2}.$$

Proof. Let u be the first node on P outside of C_f and let w be the last node of P outside of C_b. Then

$$\begin{aligned}
\overline{F}_{\min_f} &\leq \overline{F}_f(u) && \text{(by definition of } \overline{F}_{\min_f}) \\
&= \overline{f}_f^*(u) && \text{(by Theorem 10 and Lemma 8)} \\
&= \overline{f}_f(P, u) && \text{(by definition of } P) \\
&\leq \overline{f}_f(P, v) && \text{(by monotonicity of } \overline{f}_f).
\end{aligned}$$

Similarly,

$$\begin{aligned}\overline{F}_{\min_b} &\leq \overline{F}_b(w) && \text{(by definition of } \overline{F}_{\min_b}\text{)}\\ &= \overline{f}_b^*(w) && \text{(by Theorem 10 and Lemma 8)}\\ &= \overline{f}_b(P,w) && \text{(by definition of } P\text{)}\\ &\leq \overline{f}_b(P,v) && \text{(by monotonicity of } \overline{f}_b\text{).}\end{aligned}$$

Finally, Proposition 4 together with the two inequalities above imply

$$g_f(P,t) = \frac{\overline{f}_f(P,v) + \overline{f}_b(P,v)}{2} \geq \frac{\overline{F}_{\min_f} + \overline{F}_{\min_b}}{2}. \quad \square$$

Example *(cont.).* At the beginning of the fourth iteration of when DIBBS is executed on the graph depicted in Fig. 1 (see Table 1), $\overline{F}_{\min_f} = 4$ and $\overline{F}_{\min_b} = 3$. Every path from s to t must contain at least one node in $O_1 \cap O_2$, hence Theorem 11 can be used to establish that

$$C^* \geq \frac{\overline{F}_{\min_f} + \overline{F}_{\min_b}}{2} = \frac{4+3}{2} = 3.5,$$

which in turn implies $C^* \geq 4$ because every edge cost is one. Notice that this lower bound on C^* has been achieved while there still is a node in O_f with $f_f(\cdot) = 3$ and a node in O_b with $f_b(\cdot) = 3$. Thus $\overline{F}_{\min_f}$ and $\overline{F}_{\min_b}$ are able to establish a stronger lower bound than either $f_{\min_f}$ or $f_{\min_b}$.

The next theorem shows that DIBBS terminates on or before the two searches meet, which is defined to mean that a node that has been expanded in one direction is eligible to be expanded in the opposite direction, i.e., there exists a node $v \in C_d$ such that $v \in O_{d'}$ and $\overline{F}_{d'}(v) = \overline{F}_{\min_{d'}}$.[2] Note that it is not necessary to explicitly check if the two searches have met — the termination criteria automatically guarantees termination when the searches meet. Sadhukhan [16] did not analyze this aspect of the algorithm.

Theorem 12. *DIBBS terminates in a finite number of iterations with the optimal path. Furthermore, DIBBS terminates on or before the first iteration in which a node that is closed in one direction is chosen to be expanded in the opposite direction.*

Proof. First, we show that a node is expanded at most once in the forward direction and at most once in the backward direction. Let u be a node chosen to be expanded in the forward direction. Theorem 10 implies that $\overline{F}_f(u) = \overline{f}_f^*(u)$ when u is expanded, thus $\overline{F}_f(u)$ cannot decrease after u is moved from O_f to C_f. Lemma 9 states that $\overline{F}_{\min_f}$ increases monotonically, hence u will never be chosen to be expanded in the forward direction again. Consequently, if DIBBS' termination criteria is ever satisfied, it will occur within a finite number of iterations.

Second, we show that DIBBS will not terminate before an optimal path is found. Let $P^* = (v_0 = s, v_1, v_2, \ldots, v_k = t)$ be an optimal path from s to t. Suppose UB $> C^*$, i.e., an optimal path has not been found yet. Then there exists $v_i \in P^*$ such that v_i has not been expanded in either direction. Therefore $v_i \in O_f \cap O_b$ and Theorem 12 implies

$$\text{UB} > C^* = g_f\left(P^*, t\right) \geq \frac{\overline{F}_{\min_f} + \overline{F}_{\min_b}}{2},$$

which means DIBBS' termination criteria has not been satisfied.

Third, we show that the termination criteria is satisfied on or before the two searches meet. Suppose that v is the first node that is chosen to be expanded in one direction which is closed in the opposite direction. Without loss of generality, assume $v \in C_f$ and v has been chosen as the next node to expand in the backward direction. Let P be the path from s to t through v composed of the current path P_{sv} from s to v as determined by the forward search and the current path P_{vt} from v to t as determined by the backward search. We want to show that the current upper bound UB is less than or equal to the length of P. Let w be the node immediately following v on P. Lemma 9 and the fact that $v \in C_f$ imply that $\overline{F}_f(v) \leq \overline{F}_{\min_f}$. Furthermore, $\overline{F}_b(v) = \overline{F}_{\min_b}$ since v has been chosen as the next node to expand in the backward direction. Then

[2] Admittedly, this definition might be somewhat unnatural. It might be more natural to say that the two searches meet when the same node has been generated in both directions.

$$\begin{aligned}
UB &\leq G_f(w) + G_b(w) && \text{(by Theorem 10(4))}\\
&= g_f(P, w) + g_b(P, w) && \text{(by Lemma 7)}\\
&= g_f(P, t)\\
&= \frac{\overline{f}_f(P,v)+\overline{f}_b(P,v)}{2} && \text{(by Proposition 4)}\\
&= \frac{\overline{F}_f(v)+\overline{F}_b(v)}{2} && \text{(by Lemma 7)}\\
&\leq \frac{\overline{F}_{\min_f}+\overline{F}_{\min_b}}{2} && \text{(because } \overline{F}_f(v) \leq \overline{F}_{\min_f} \text{ and } \overline{F}_b(v) = \overline{F}_{\min_f}\text{).}
\end{aligned}$$

Therefore DIBBS' termination criteria has been satisfied and v will not be expanded in the backward direction.

Finally, Theorem 11 shows that when the termination criteria has been satisfied, it is not possible to find a shorter path that goes through the open nodes. □

We now provide sufficient conditions under which a node will not be expanded by DIBBS. This requires that we specify how the search direction is chosen at each iteration. The *Best First Direction (BFD)* rule selects the search direction by choosing the direction that has the smallest $\overline{F}_{\min_d}$.

Theorem 13. *If the BFD rule is used with DIBBS, then*

1. *An optimal path will be discovered while* $\overline{F}_{\min_f} \leq C^*$ *or* $\overline{F}_{\min_b} \leq C^*$ *(or both).*
2. *If* $\overline{f}^*_f(v) > C^*$*, then* v *is not expanded in the forward direction.*
3. *If* $\overline{f}^*_b(v) > C^*$*, then* v *is not expanded in the backward direction.*
4. *If* $\min\left[\overline{f}^*_f(v), \overline{f}^*_b(v)\right] > C^*$*, then* v *is not expanded in either direction.*

Proof. Let $P = (v_0 = s, v_1, v_2, \ldots, v_k = t)$ be the first shortest path from s to t discovered by DIBBS and v_i be a node on P. Then

$$\begin{aligned}
C^* &= \frac{\overline{f}_f(P,v_i)+\overline{f}_b(P,v_i)}{2} && \text{(by Proposition 4)}\\
&= \frac{\overline{f}^*_f(v_i)+\overline{f}^*_b(v_i)}{2} && \text{(by Proposition 2).}
\end{aligned}$$

Consequently, either $\overline{f}^*_f(v_i) \leq C^*$ or $\overline{f}^*_b(v_i) \leq C^*$. For the BFD rule, this implies that every node on P will be expanded while $\overline{F}_{\min_f} \leq C^*$ or $\overline{F}_{\min_b} \leq C^*$, which in turn implies that P will be discovered while $\overline{F}_{\min_f} \leq C^*$ or $\overline{F}_{\min_b} \leq C^*$.

To prove (2), let v be a node such that $\overline{f}^*_f(v) > C^*$. Let u be the last node on P such that $\overline{f}^*_f(u) \leq C^*$. Such a node must exist because $\overline{f}^*_f(s) = 2g^*_f(s) + h_f(s) - h_b(s) = h_b(s) \leq C^*$. Then u must have been expanded before v was expanded in the forward direction. Let w be the node that immediately follows u on P. If such a node does not exist, i.e., u is the last node on P, then let $w = u$. Then $\overline{f}^*_f(w) \geq C^*$ by the choice of u and w. Thus

$$\begin{aligned}
C^* &= \frac{\overline{f}_f(P,w)+\overline{f}_b(P,w)}{2} && \text{(by Proposition 4)}\\
&= \frac{\overline{f}^*_f(w)+\overline{f}^*_b(w)}{2} && \text{(by Proposition 2)}\\
&\geq \frac{C^*+\overline{f}^*_b(w)}{2}.
\end{aligned}$$

Consequently, $\overline{f}^*_b(w) \leq C^*$. Hence, w must have been expanded before v was expanded in the forward direction. Therefore P will be discovered and UB will be set equal to C^* before v is expanded in the forward direction.

Now suppose that v is selected to be expanded in the forward direction. The BFD rule together with the hypothesis $\overline{f}^*_b(v) > C^*$ imply $\min\left[\overline{F}_{\min_f}, \overline{F}_{\min_b}\right] = \overline{f}^*_f(v) > C^*$, which yields

$$\begin{aligned}
UB &= C^*\\
&< \min\left[\overline{F}_{\min_f}, \overline{F}_{\min_b}\right]\\
&\leq \frac{\overline{F}_{\min_f} + \overline{F}_{\min_b}}{2}.
\end{aligned}$$

Therefore the algorithm's termination criteria has been satisfied and v will not be expanded in the forward direction.

Part (3) is proved similarly. Part (4) follows immediately from parts (2) and (3). □

We are now in a position to show that DIBBS with BFD is theoretically capable of expanding fewer nodes than either FA* or BA*. Begin by observing that $\overline{f}^*_d(v) = f^*_d(v) + g^*_d(v) - h_{d'}(v) \geq f^*_d(v)$ due to the consistency of h_f and h_b. Next, observe

that FA* never expands a node v such that $f_f^*(v) > C^*$, hence neither does the forward search of DIBBS because $\overline{f}_f^*(v) > C^*$ whenever $f_f^*(v) > C^*$. On the other hand, FA* must expand every node v such that $f_f^*(v) < C^*$, but the forward search of DIBBS does not have to expand all such nodes. In particular, if $\overline{f}_f^*(v) > C^* > f_f^*(v)$, then FA* must expand v but the forward search of DIBBS does not. Of course, by no means does this prove that DIBBS must expand fewer nodes in total (sum of both search directions) than either FA* or BA*. In fact, such a statement is false. But it proves that it is theoretically possible. The computational experiments in Section 5 will demonstrate that it is possible in practice as well.

Example *(cont.).* Let $G = (V, E)$ be the graph depicted in Fig. 1. Then

$$\min\left[\overline{f}_f^*(b), \overline{f}_b^*(b)\right] > C^* \text{ and } \min\left[\overline{f}_f^*(c), \overline{f}_b^*(c)\right] > C^*,$$

thus both of these nodes satisfy Theorem 13(4), hence they will not be expanded in either direction by DIBBS. But both nodes must be expanded by FA* and BA*. DIBBS expands six nodes when it is applied to G (s, t, a, d, x, z) (see Table 1). FA* must expand at least seven nodes (s, a, b, c, x, y, z) and might also expand node d, depending on the tie-breaking rule. Similarly, BA* also must expand at least seven nodes (s, d, c, b, z, y, z) and might also expand node a, depending on the tie-breaking rule.

Sadhukhan [16] provided the following analysis of his BAE* algorithm. Let

$$V_f = \left\{v : \exists \text{ path } P \text{ from } s \text{ to } v \ni \overline{f}_f(u) \leq 2C^* - h_f(s) \quad \forall u \in P\right\}$$
$$V_b = \left\{v : \exists \text{ path } P \text{ from } v \text{ to } t \ni \overline{f}_b(u) \leq 2C^* - h_b(t) \quad \forall u \in P\right\}.$$

He proved that BAE* expands at most $|V_f| + |V_b|$ nodes by showing that BAE* never expands a node in the forward direction outside of V_f and never expands a node in the backwards direction outside of V_b. For the ease of exposition, assume that $h_f(s) = h_b(t)$. This means that BAE* never expands a node v such that

$$\begin{aligned}\min\left[\overline{f}_f(v), \overline{f}_b(v)\right] &> 2C^* - h_f(s)\\ &= C^* + \left(C^* - h_f(s)\right).\end{aligned}$$

Consequently, this bound is not as tight as the bound given in Theorem 13, except in the case $C^* = h_f(s)$, i.e., except when the heuristic perfectly estimates the distance between s and t. For example, we have just shown in the preceding example that Theorem 13 guarantees that neither node b nor node c in the graph depicted in Fig. 1 will be expanded by DIBBS, whereas Sadhukhan's bound does not provide such a guarantee for these two nodes, because

$$\min\left[\overline{f}_f(b), \overline{f}_b(b)\right] = 5 \leq 2C^* - h_f(s) = 2(4) - 2 = 6$$
$$\min\left[\overline{f}_f(c), \overline{f}_b(c)\right] = 5 \leq 2C^* - h_f(s) = 2(4) - 2 = 6.$$

If heuristics are not used in DIBBS, i.e., $h_f(u) = h_b(u) = 0 \forall u \in V$, then DIBBS reverts to Dijkstra's bidirectional search algorithm, which also is call brute-force bidirectional search. In this case, it will avoid expanding the same nodes that Djikstra's BS algorithm avoids, as noted in the following corollary.

Corollary 14. *If $h_f(u) = h_b(u) = 0$ for all $u \in V$, then DIBBS with BFD does not expand nodes such that* $\min\left[g_f^*(v), g_b^*(v)\right] > C^*/2$.

Proof. From Proposition 1 and the hypothesis, $\overline{f}_d^*(v) = 2g_d^*(v) + h_d(v) - h_{d'}(v) = 2g_d^*(v) + 0 - 0 = 2g_d^*(v)$. Theorem 13 states that the bidirectional algorithm with BFD does not expand v if

$$\begin{aligned}C^* &< \min\left[\overline{f}_f^*(v), \overline{f}_b^*(v)\right]\\ &= \min\left[2g_f^*(v), 2g_b^*(v)\right].\end{aligned}$$

Or equivalently, $\min\left[g_f^*(v), g_b^*(v)\right] > C^*/2$. □

The new bidirectional algorithm permits us to exclude nodes that have

$$\min\left[\overline{f}_f(P, v), \overline{f}_b(P, v)\right] > C^*$$

without knowing the optimal value C^* in advance. It accomplishes this by using best first search with $\overline{f}_f, \overline{f}_b$ as the measure of best. This is similar to the improvement in moving from a unidirectional shortest path algorithm without heuristics to a unidirectional shortest path algorithm that uses heuristics. Without the heuristics, the algorithm can exclude nodes whose distance is greater than C^*, without knowing C^* in advance. With the heuristics, it can exclude nodes with $f(v) > C^*$, without knowing C^* in advance.

Several comments regarding DIBBS are in order. The algorithm was presented in such a way as to make the analysis of the algorithm as simple as possible. However, it is possible to make some practical improvements. First, O_f and O_b can be initialized as $O_f = \{s\}$ and $O_b = \{t\}$, as is common in other BHS algorithms. In many situations, it is completely impractical to explicitly place every node in O_f and O_b. Nodes can be added to O_d the first time their distance labels are modified in the expand functions. Second, it may be possible to prune some nodes as soon as they are generated, thus avoiding having to use memory to store them. This was called early goal testing by Felner [8] and is a common practice in recent BHS algorithms. For example, for our computational experiments in Section 5, we used $f_f(w) \geq UB$ to prune nodes as they were generated. Third, it may be possible to prune some nodes when they are selected for expansion. For example, for our computational experiments, we also used $f_f(w) \geq UB$ to prune nodes when they were selected for expansion. Fourth, additional termination criteria, such as $\min\left\{f_f(v) : v \in O_f\right\} \geq UB$ or $\min\left\{G_f(v) : v \in O_f\right\} + \min\{G_b(v) : v \in O_b\} \geq UB$ can be employed.

We now compare DIBBS to other front-to-end BHS algorithms. One advantage that DIBBS has over most BHS algorithms is that it terminates on or before the two searches meet. BHPA [15] permitted nodes to be expanded twice, once in each direction. This resulted in the two searches passing through each other (see [12]). Kwa's BS* algorithm [14] overcame this problem, but it permitted a node to be expanded more than once in the same direction. Auer and Kaindl's algorithms [1] both suffered from this same property. More recently, Holte et al. [11] achieved this same advantage via an entirely different approach. Of course, DIBBS's biggest advantage is that it dynamically improves the bounds with virtually no additional computational effort. As discussed above, the improved bounds permit it to exclude some of the nodes that must be expanded by A*. This results in an algorithm that is capable of breaking through the theoretical barrier that Kaindl and Kainz [12] proved about BHPA, namely, that if all the f-values are distinct, then BHPA must expand at least as many nodes as either FA* or BA* expands.

It is worthwhile to compare DIBBS directly to the algorithms developed by Auer and Kaindl [1], since their algorithms and DIBBS are both built on the foundation laid by Kaindl and Kainz [12]. As discussed in Section 3, Auer and Kaindl defined

$$F_f(v) = \max\left(f_f(v), f_{\min_b} + g_f(v) - h_b(v)\right),$$

where $f_{\min_b}$ is the minimum value of f_b among all open nodes in the backward direction ($F_b(v)$ defined similarly). Their Max-BS* algorithm uses F_d as the priority function in one of the search directions. It does not use the improved bounds for pruning in either direction. Their BiMax-BS$^*_F$ algorithm uses F_d for pruning in both directions, but does not use it as a priority function. Their computational results for the sliding tile puzzle problem indicated that both algorithms performed better than BS*, meaning that using the improved bounds as the priority function is helpful and using them for pruning is helpful. The primary improvement of DIBBS is that it uses the improved bounds in both the priority function and for pruning. Additionally, DIBBS with BFD is able to exclude nodes with $\min\left[\overline{f}^*_f(v), \overline{f}^*_b(v)\right] > C^*$, without knowing C^* in advance, i.e., it is able to prune such nodes before the optimal path is found. In contrast, BiMax-BS$^*_F$ cannot use the improved lower bound for pruning until the first path from s to t has been discovered, and the full strength of the bounds cannot be used until the optimal path is found. Another advantage of DIBBS is that whenever a shorter path is discovered from s to t, MAX-BS* and BiMAX-BS$^*_F$ retroactively removed nodes from the search, which consumes additional time, whereas DIBBS does not need to do so. One final advantage of DIBBS is that it terminates on or before the two searches meet, whereas Max-BS* and BiMax-BS$^*_F$ continue to process nodes until one of the sets of open nodes is empty.

5. Computational experiments

This section presents computational experiments with DIBBS on the pancake problem, the sliding tile puzzle, and on the topspin problem. Note that Sadhukhan [16] only reported results for BAE* for the sliding tile puzzle. The primary purpose of these experiments is a proof of concept to demonstrate the following claims in practice.

1. DIBBS is capable of solving certain problems while expanding fewer nodes than either FA* or BA*.
2. DIBBS can overcome Barker and Korf's [2] second conclusion that with a strong heuristic, BHS will expand more nodes than a unidirectional heuristic search.
3. Barker and Korf conjectured that the contribution of Kaindl and Kainz's BHS algorithm was to avoid generating nodes with f cost equal to the cost of an optimal path, i.e., Kainld and Kainz's algorithm performed better than A* merely because it had a better tie-breaking rule which permitted it to find the optimal path sooner. DIBBS is capable of solving problems while expanding fewer nodes with $f(v) < C^*$ than A*.
4. DIBBS is capable of solving certain problems while expanding fewer nodes than the theoretical minimum for DXBB BHS algorithms. Hence for certain problems it is capable of outperforming any DXBB algorithm, such as MM, MM$_\epsilon$, GBFHS, NBS, MT, DVCBS$_F$, and DVCBSF$_A$.

Furthermore, DIBBS is compared to MM_ϵ by Holte et al. [10,11,19] and to GBFHS by Barley et al. [3], which are two state-of-the-art bidirectional algorithms. MM_ϵ is designed in such a way that its forward search never expands a node u with $g_f^*(u) > (C^* - \epsilon)/2$ and its backward search never expands a node v with $g_b^*(v) > (C^* - \epsilon)/2$, where ϵ is the smallest cost of an edge. As a result, the two searches meet in the middle, as measured by each node's distance from s or t. GBFHS controls where the forward and backward searches meet by expanding one level of nodes at a time. Let $fLim = \min\left(f_{\min_f}, f_{\min_b}\right)$. At the beginning of a level, a split function is used to determine two limits, $gLim_f \geq 0$ and $gLim_b \geq 0$, such that $gLim_f + gLim_b = fLim - \epsilon + 1$. While expanding this level, GBFHS limits the expansion of nodes in the forward direction to nodes such that $g_f^*(u) < gLim_f$ and $f_f(u) \leq fLim$ and in the backward direction to nodes such that $g_b^*(v) < gLim_b$ and $f_b(v) \leq fLim$. By means of an appropriate definition of the split function, GBFHS can limit the search to the forward direction, similar to FA*, or the backward direction, similar to BA*, or to meet in the middle, or to meet at predetermined values of g_f and g_b. The split function for the computational results presented in [3] was not specified. Via a private communication, the authors stated that the split function was chosen to imitate meet in the middle. The split function used in this paper does the same by enforcing $\left|gLim_f - gLim_b\right| \leq 1$. An additional tie-breaking rule was used when $gLim_f = gLim_b$: if $\left|O_f\right| \leq |O_b|$, then $gLim_f$ is incremented, otherwise $gLim_b$ is incremented.

5.1. The pancake problem

Given a stack of n pancakes of distinct diameters, the goal of the pancake problem is to sort them from the smallest to the largest. The only sorting operation that is permitted is to insert a spatula between two pancakes and then flip the stack on the spatula. The objective is to minimize the number of flips needed to sort the pancakes. Given a permutation π of the integers 1 through n, a prefix reversal of length k is the sequence obtained from π by reversing the order of the first k elements in π. The prefix reversal sorting problem is to find the minimum number of reversals needed to sort the sequence into increasing order. The pancake problem, which is NP-hard [4], is directly equivalent to the prefix reversal sorting problem. One thousand random instances were generated for each value of $n = 10, 20, 30, 40$ for the experiments below.

Helmert [9] developed the GAP heuristic for the pancake problem. Given a permutation π of the integers 1 through n, a *gap* occurs at location i if π_i and π_{i+1} are not consecutive integers. An additional gap occurs at location n if $\pi_n \neq n$. The GAP heuristic for π is the total number of gaps. This is an extremely strong heuristic. For example, out of the 1,000 randomly generated instances with $n = 40$, the GAP heuristic equaled C^* for 345 of the instances, differed by one for 645 of the instances, and differed by two for 10 of the instances. It did not differ by more than two for any of the instances. One of the primary reasons we selected the pancake problem for our computational experiments was the availability of an extremely strong heuristic. This strong heuristic permits us to refute the claim that BHS cannot outperform FA* or BA* when a strong heuristic is used. A secondary reason for selecting the pancake problem is that the GAP heuristic can be easily weakened, as shown by Holte et al. [10], by ignoring the gaps involving the X smallest pancakes. The resulting heuristic is called GAP-X, where GAP-0 is the same as the original GAP heuristic. We use GAP-1, GAP-2, and GAP-3 to demonstrate how DIBBS performs when given a weaker heuristic.

The computational experiments are presented in Table 2. In this table, n is the number of pancakes and the columns labeled FA*, BA*, Min(FA*,BA*), MM_ϵ, GBFHS, and DIBBS contain the average number of nodes expanded by each of these algorithms. If the entry is a percentage, then this is the percentage of the 1,000 instances that were successively solved before running out of memory. An entry of "DNR" indicates that we did not run the algorithm because it was estimated that very few of the instances would complete before running out of memory. The Min(FA*, BA*) algorithm is the algorithm obtained by running both FA* and BA* and choosing the better one for each problem, i.e., it is the algorithm obtained if we had an oracle that could choose the better direction in advance. The table is divided into sections based on which heuristic was used. For each row, the fewest number of node expansions is in bold. For these experiments, the direction for each iteration of DIBBS was chosen using the cardinality rule, namely that if $\left|O_f\right| \leq |O_b|$, then the forward direction is selected; otherwise the backward direction is selected.

The results presented in Table 2 demonstrate that DIBBS is capable of solving problems with fewer node expansions than either FA* or BA*. For example, for $n = 40$ with the GAP-0 heuristic, FA* expanded 3.7 times as many nodes as DIBBS. For weaker heuristics, the ratio is larger. For example, for $n = 10$ with the GAP-3 heuristic, FA* expanded 58.3 times as many nodes as DIBBS. Furthermore, even if an oracle is available to choose the better direction in advance, DIBBS still expands fewer nodes. For example, for $n = 40$ with the GAP-0 heuristic, Min(FA*, BA*) expanded twice as many nodes as DIBBS and for $n = 10$ with the GAP-3 heuristic, Min(FA*, BA*) expanded 18.9 times as many nodes as DIBBS. The results also demonstrate that DIBBS is capable of solving problems with fewer node expansions than either MM_ϵ or GBFHS. For example, for $n = 30$ with the GAP-0 heuristic, MM_ϵ expanded 29.8 times as many nodes as DIBBS and GBFS expanded 2.6 times as many nodes as DIBBS.

5.2. The sliding tile puzzle

The sliding tile puzzle has been extensively used as a test problem for heuristic search algorithms, both unidirectional and bidirectional. This section presents computational results for the 15-puzzle on the 100 instances created by Korf [13] using the Manhattan heuristic.

Table 2
Computational results for the pancake problem. The columns labeled FA*, BA*, Min(FA*, BA*), MM_ϵ, GBFHS, and DIBBS contain the average number of nodes expanded. If the entry is a percentage, then this is the percentage of the 1,000 instances that were successively solved before running out of memory. An entry of "DNR" indicates that we did not run the algorithm because of the estimated difficulty.

Initial UB = ∞						
n	FA*	BA*	Min(FA*, BA*)	MM_ϵ	GBFHS	DIBBS
GAP-0						
10	54	53	43	76	33	**30**
20	1,304	1,298	904	6,905	877	**527**
30	13,092	12,461	8,019	127,634	11,042	**4,286**
40	92,309	88,234	56,702	96.9%	70,411	**24,914**
GAP-1						
10	509	572	387	559	274	**160**
20	152,365	156,769	92,699	881,162	89,588	**32,384**
30	99.8%	86.1%		61.4%	92.9%	**1,089,620**
40	53.3%	23.2%		DNR	DNR	99.1%
GAP-2						
10	8,201	3,331	3,221	1,681	929	**479**
20	87.2%	97.4%		56.6%	95.2%	**548,448**
30	DNR	DNR		DNR	DNR	81.1%
GAP-3						
10	53,033	17,638	17,165	2,433	1,655	**909**
20	DNR	45.8%		DNR	DNR	99.9%

Table 3
Computational results for the 15-puzzle using the MD heuristic. The columns contain the average number of node expansions for the 100 instances.

Initial UB = ∞						
IDA*	MM_ϵ	NBS	GBFHS	$DVCBS_F$	$DVCBS_A$	DIBBS
184,336,714	13,162,312	12,851,889	12,507,393	11,669,720	11,933,791	**1,603,867**

The computational results are presented in Table 3. FA* and BA* required too much memory to be solved, so results for IDA* are reported instead. The table also includes NBS, $DVCBS_F$, and $DVCBS_A$, since results for these algorithms on the 15-puzzle have been published. The results for MM_ϵ and NBS are as reported in [5], the results for GBFHS are as reported in [3], and the results for $DVCBS_F$ and $DVCBS_A$ are as reported in [20]. For these experiments, the direction for each iteration of DIBBS was chosen using the cardinality rule with $\overline{f}$-leveling, meaning that once a direction, say d, has been chosen using the cardinality rule, then all of the nodes $u \in O_d$ with $\overline{f}_d(u) = \overline{f}_{\min_d}$ are expanded before switching directions. Furthermore, when choosing the next node to be expanded, ties were broken in favor of nodes with larger g_d. This tie-breaking rule permits DIBBS to find the optimal path sooner, on average.

The results presented in Table 3 clearly demonstrate that DIBBS dominates the other algorithms. For unidirectional search, IDA* expanded 115 times as many as nodes as DIBBS. For bidirectional search, $DVCBS_F$, DIBBS' closest competitor, expanded 7.3 times as many nodes as DIBBS.

5.3. The topspin problem

Given two positive integers N and K and a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$ of the integers 1 through N, the goal of the topspin problem is to sort the integers in increasing order by repeatedly performing two operations. The first operation is to rotate the permutation at position i resulting in a new permutation $\pi' = (\pi_i, \pi_{i+1}, \ldots, \pi_N, \pi_1, \pi_2, \ldots, \pi_{i-1})$. The second operation is to reverse the order of the first K consecutive elements of π, resulting in a new permuation $\pi' = (\pi_K, \pi_{K-1}, \ldots, \pi_1, \pi_{K+1}, \pi_{K+2}, \ldots, \pi_N)$. Only the second operation is counted as a move, i.e., the cost of the first operation is zero. For these experiments, a single pattern database of M consecutive positions was used (see [22]). This single pattern database was used N times, once for positions $(1, 2, \ldots, M)$, once for $(2, \ldots, M+1)$, etc. The maximum of the N lookups was used as the heuristic value. The overall quality of the heuristic can be varied by varying M. Smaller values of M result in a weaker heuristic, while larger values of M result in a stronger heuristic. One hundred random instances were generated for $N = 16$ and $N = 18$, with K set equal to 4.

The computational experiments are presented in Table 4. In this table, the columns labeled FA*, BA*, Min(FA*, BA*), MM_ϵ, GBFHS, and DIBBS contain the average number of nodes expanded by each of these algorithms. If the entry is a percentage, then this is the percentage of the 100 instances that were successively solved before running out of memory. For these experiments, the direction for each iteration of DIBBS was chosen using the cardinality rule with $\overline{f}$-leveling. The table also contains a column labeled "$\% \leq C^*/2$," which is the percentage of expanded nodes with $g^*_f(\cdot) \leq C^*/2$. Barker and Korf [2]

Table 4
Computational results for the topspin problem. The columns labeled FA*, BA*, Min(FA*, BA*), MM_ϵ, GBFHS, and DIBBS contain the average number of nodes expanded. If the entry is a percentage, then this is the percentage of the 100 instances that were successively solved before running out of memory.

Initial UB $= \infty$								
N	M	FA*% $\leq C^*/2$	FA*	BA*	Min(FA*, BA*)	MM_ϵ	GBFHS	DIBBS
16	5	71.0%	7,033,747	7,847,832	5,248,213	1,232,463	1,150,201	**1,027,882**
16	6	97.2%	942,035	1,169,709	805,372	821,916	569,561	**311,115**
16	7	99.6%	130,260	137,172	111,914	426,603	95,324	**49,071**
16	8	99.1%	18,566	18,206	15,675	91,085	16,010	**9,724**
16	9	97.1%	3,611	3,135	2,778	15,340	3,354	**1,990**
18	7	99.0%*	91%	92%		12,516,576	4,807,724	**2,292,688**
18	8	99.7%	1,023,621	944,035	815,161	2,253,649	673,666	**335,657**
18	9	99.5%	122,971	122,055	104,721	459,145	88,315	**53,995**

* Computed for the 91 instances that were solved.

suggested this as a measure of the strength of a heuristic. If this percentage is greater than one-half, then the heuristic is deemed to be strong, otherwise it is deemed to be weak. By this measure, all the heuristics presented in the table are strong, but clearly $M = 5$ for $N = 16$ is weaker than the others.

Similar to the pancake problem, the results presented in Table 4 demonstrate that DIBBS is capable of solving problems with fewer node expansions than either FA* or BA*. For example, for $N = 18$ with $M = 9$, FA* expanded 2.3 times as many nodes as DIBBS. For weaker heuristics, the ratio is larger. For example, for $N = 16$ with $M = 5$, FA* expanded 6.8 times as many nodes as DIBBS. Furthermore, even if an oracle is available to choose the better direction in advance, DIBBS still expands fewer nodes. For example, for $N = 18$ with $M = 9$, Min(FA*, BA*) expanded 1.9 times as many nodes as DIBBS and for $N = 16$ with $M = 5$, Min(FA*, BA*) expanded 5.1 times as many nodes as DIBBS. The results also demonstrate that DIBBS is capable of solving problems with fewer node expansions than either MM_ϵ or GBFHS. For example, for $N = 18$ with $M = 9$, MM_ϵ expanded 8.5 times as many nodes as DIBBS and GBFS expanded 1.6 times as many nodes as DIBBS. For $N = 18$ with $M = 7$, MM_ϵ expanded 5.5 times as many nodes as DIBBS and GBFS expanded 2.1 times as many nodes as DIBBS.

6. Minimum number of node expansions

One way to establish the quality of an algorithm is to prove that it possesses desirable theoretical properties. This was done in Theorem 12, where it was shown that DIBBS terminates on or before the two searches meet, and Theorem 13, where it was shown that it is theoretically possible for DIBBS to expand fewer nodes that either FA* or BA*. A second way to establish the quality of an algorithm is to compare it directly to existing algorithms via computational experiments, as was done in Section 5. A third way for expansion-based search algorithms is to compare the number of nodes expanded by the algorithm to the theoretical minimum number of node expansions that are necessary to verify the optimality of the solution. This is the approach that will be taken in this section.

The heuristics h_f and h_b are *admissible* if $h_f(v) \leq g_b^*(v)$ and $h_b(v) \leq g_f^*(v)$ for all $v \in V$. If h_f and h_b are consistent, then they are admissible if and only if $h_f(t) = 0$ and $h_b(s) = 0$. A heuristic search algorithm is admissible if it is guaranteed to return an optimal solution whenever the heuristics are admissible.

Dechter and Pearl [6] proved that under some reasonable assumptions A* is optimally efficient (terminology from [7]) among unidirectional search algorithms. In particular, they proved that every admissible unidirectional search algorithm must expand every node v such that $f(v) < C^*$. Given a consistent heuristic, A* expands each such node exactly once, consequently, no admissible unidirectional search can perform better than A* in this regard. The theory does not address nodes such that $f(v) = C^*$. Other unidirectional algorithms might expand fewer nodes with $f(v) = C^*$ than A*.

Eckerle et al. [7] clarified the assumptions made by Dechter and Pearl and extended the assumptions to the bidirectional setting. They defined a search algorithm to be a Deterministic Expansion-based Black Box (DXBB) algorithm if it is deterministic, expands only previously generated nodes, and only has black-box access to the expand, heuristic, and cost functions. For front-to-end BHS algorithms, they proved that if a pair of nodes $(u, v) \in V$ satisfies

1. $f_f(u) < C^*$
2. $f_b(v) < C^*$
3. $g_f^*(v) + g_b^*(v) < C^*$,

then any admissible DXBB front-to-end BHS algorithm must expand at least one of u or v. Such a pair of nodes is called a *Must Expand Pair (MEP)*. This result does not directly determine the minimum number of nodes that must be expanded, because it does not specify which of the two nodes should be expanded or whether both nodes must be expanded. Chen et al. [5] answered this question by defining $lb(u, v) = \max\left(f_f(u), f_b(v), g_f^*(v) + g_b^*(v)\right)$ and building a bipartite graph G_{MX}, called the must-expand graph, that contains two nodes, u_1 and u_2 (on opposite sides of the bipartition) for every node $u \in V$. Two nodes u_1 and v_2 in G_{MX} are adjacent if and only if $lb(u, v) < C^*$. They then proved that a minimum

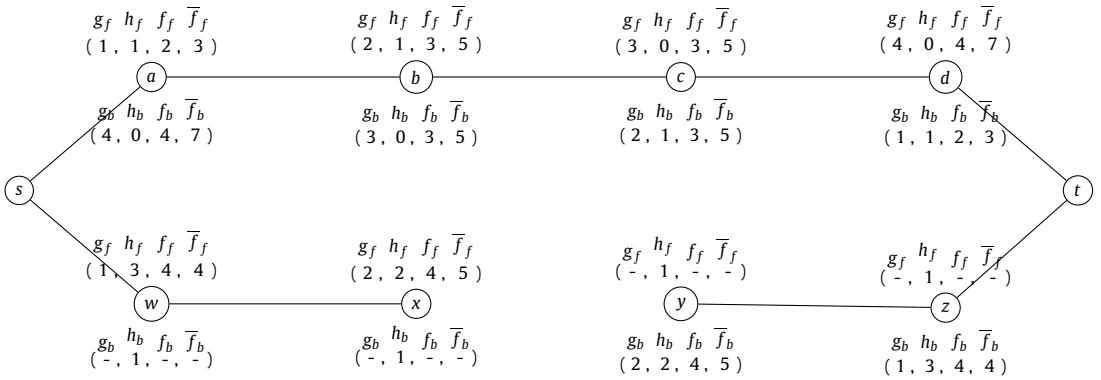

Fig. 6. A graph with unit costs. The optimal path is (s, a, b, c, d, t) and $C^* = 5$.

vertex cover of G_{MX} corresponds to the smallest set of nodes that must be expanded by any admissible DXBB front-to-end BHS algorithm. Shaham et al. [17] showed that nodes in G_{MX} can be clustered together. Using this insight, they proved that the minimum vertex cover can be computed in time linear in the size of G_{MX}. Shaham et al. [18] proved that if the smallest cost of an edge is known, then $lb_\epsilon(u, v) = \max\left(f_f(u), f_b(v), g_f^*(v) + g_b^*(v) + \epsilon\right)$ can be used in place of $lb(u, v)$, resulting in a graph G_{MX_ϵ} that contains fewer edges than G_{MX}, which might result in a smaller minimum vertex cover. The minimum vertex cover of G_{MX} $\left(G_{MX_\epsilon}\right)$is denoted MVC (MVC$_\epsilon$).

DIBBS is not a DXBB algorithm, so it is possible for DIBBS to solve a problem while expanding fewer nodes than in the MVC. This is demonstrated in the following example and in the computational results below. This phenomenon occurs because the analysis in [5,7,17] implicitly assumes that DXBB algorithms only use h_f when expanding nodes in the forward direction and only use h_b when expanding nodes in the backward direction. DIBBS violates this assumption by using h_f and h_b in both directions to dynamically improve the bounds, hence DIBBS is not a DXBB algorithm and is not subject to the associated theoretical minimum. Furthermore, DIBBS assumes that the heuristics are consistent, and exploits the consistency to dynamically improve the bounds, whereas DXBB algorithms do not assume that the heuristics are consistent, with the one exception of the MT$_{\text{CON}}$ algorithm developed by Shaham et al. [18]. It is an area of future research to develop an MEP theory for DIBBS.

Example. Let $G = (V, E)$ be the graph depicted in Fig. 6. Nodes s and t are left unlabeled because some of the values depend on which path is traversed. Let $h_f(s) = 2$, $h_b(s) = 0$, $h_f(t) = 0$, and $h_b(t) = 2$. The optimal path is (s, a, b, c, d, t) and $C^* = 5$. Suppose DIBBS is applied to G with the BFD rule. DIBBS will begin by expanding nodes s, t, a, d, w, and z. At this point, nodes b, c, x, and y are eligible to be expanded. Suppose node b is chosen to be expanded next. Then the optimal path will be discovered and the algorithm will terminate without expanding c, x, or y because $\left(\overline{F}_{\min_f} + \overline{F}_{\min_b}\right)/2 = (5+5)/2 \geq$ UB $= 5$. Nodes x and y are an MEP because $f_f(x) = 4 < C^*$, $f_b(y) = 4 < C^*$, and $g_f^*(x) + g_b^*(y) = 4 < C^*$, so at least one of them must be expanded by any DXBB algorithm, but neither one was expanded by DIBBS. That DIBBS requires consistent heuristics can be seen by adding the edge (x, y) to G with cost zero. The heuristics are no longer consistent because $h_f(x) \nleq c(x, y) + h_f(y)$, but they are still admissible. In this scenario, the optimal path is (s, w, x, y, z, t) and $C^* = 4$, but this path will not be found because DIBBS did not expand either x or y.

As mentioned above, the analysis of unidirectional algorithms does not address nodes with $f(v) = C^*$. In a similar manner, the analysis of the minimum number of nodes that must be expanded by front-to-end BHS algorithms does not address pairs of nodes with $lb(u, v) = C^*$. As such, the theory addresses the minimum number of nodes that must be expanded to verify optimality, but it does not address the minimum number of nodes that must be expanded to find the optimal solution. A BHS algorithm, call it Algorithm 1, might achieve perfect success at expanding as few nodes as possible to verify optimality (i.e., it expands a minimum vertex cover of G_{MX}) and yet might need to expand more nodes to find the optimal path. On the other hand, a different BHS algorithm, call it Algorithm 2, might expand more nodes than required for a minimum vertex cover of G_{MX} but might expand fewer nodes in total than Algorithm 1 because it expands fewer nodes to find the optimal path.

The computational results for the pancake problem are presented in Table 5. There are three differences between Tables 2 and 5. First, in Table 5 the optimal value C^* is supplied as the initial upper bound to FA*, BA*, Min(FA*, BA*), MM$_\epsilon$, GBFHS, and DIBBS. Therefore these algorithms do not need to find the optimal solution; they only need to verify optimality. Second, Table 5 includes the size of the minimum vertex covers MVC and MVC$_\epsilon$. Third, the last column displays the percentage of nodes expanded by DIBBS when C^* is supplied as the initial upper bound relative to the number of nodes expanded when the initial upper bound is infinity. I.e., it is the percentage of nodes needed to verify optimality relative to the number of nodes needed to find and verify optimality.

Table 5
Minimum number of expansions for the pancake problem. The columns labeled FA*, BA*, Min(FA*, BA*), MM_ϵ, GBFHS, and DIBBS contain the average number of nodes expanded. If the entry is a percentage, then this is the percentage of the 1,000 instances that were successively solved before running out of memory. An entry of "DNR" indicates that we did not run the algorithm because of the estimated difficulty.

Theoretical minimum number of expansions						Initial upper bound = C^*			
n	FA*	BA*	Min(FA*, BA*)	|MVC|	$\|MVC_\epsilon\|$	MM_ϵ	GBFHS	DIBBS	
								Expand	%
GAP-0									
10	**8**	**8**	6	6	6	9	10	**8**	27%
20	165	165	121	121	121	196	196	**138**	26%
30	1,143	1,273	974	974	972	1,406	1,407	**1,097**	26%
40	7,458	7,227	5,699	5,696	5,673	8,361	8,307	**6,502**	26%
GAP-1									
10	509	278	275	257	209	254	236	**156**	98%
20	152,365	46,117	45,815	45,632	44,794	87,604	80,178	**31,130**	96%
30	99.9%	1,290,999		99.9%	99.9%	99.9%	3,583,199	**922,566**	85%
40	81.5%	99.4%		DNR	DNR	DNR	DNR	99.5%	
GAP-2									
10	8,201	1,990	1,989	1,339	816	1,178	869	**422**	88%
20	99.4%	1,572,413		99.2%	99.2%	9,384,541	7,157,967	**472,212**	86%
30	DNR	88.1%		DNR	DNR	DNR	DNR	94.8%	
GAP-3									
10	53,033	11,653	11,653	3,278	1,540	1,864	1,553	**856**	94%
20	DNR	96.3%		DNR	DNR	DNR	DNR	99.9%	

Table 5 shows that when GAP-1, GAP-2, or GAP-3 is used, DIBBS is capable of verifying optimality with fewer node expansions than the theoretical minimum. For example, for $n = 10$ with the GAP-2 heuristic, DIBBS was able to verify optimality with 422 node expansions, on average, whereas $|MVC| = 1,339$ and $|MVC_\epsilon| = 816$. In fact, DIBBS was able to find and verify optimality for these same problems with 479 node expansions (see Table 2), on average, which is less than the theoretical minimum number of expansions needed just to verify optimality. NBS and DVCBS are DXBB algorithms, hence neither can expand fewer nodes than $|MVC_\epsilon|$. Consequently, although NBS and DVCBS were not included in our computational experiments for the pancake problem (Tables 2 and 5), neither algorithm can expand fewer nodes than DIBBS when GAP-1, GAP-2, or GAP-3 is used.

Table 5 shows that for the stronger heuristics, GAP-0 and GAP-1, the number of nodes expanded by Min(FA*, BA*) is nearly identical to |MVC|. This indicates that for DXBB algorithms, the best strategy when GAP-0 or GAP-1 is used is to only expand nodes in one direction, i.e., unidirectional dominates bidirectional search. This is in keeping with Barker and Korf's [2] conclusion, "With a strong heuristic, a bidirectional heuristic search expands more nodes than a unidirectional heuristic search." This conclusion is violated by DIBBS, as Table 5 shows that DIBBS expands fewer nodes to verify optimality than Min(FA*, BA*) when GAP-1 is used and Table 2 shows that DIBBS expands fewer nodes to find and verify optimality than Min(FA*, BA*) when GAP-0 is used. This is discussed in greater detail in Section 7. For the weaker heuristics, GAP-2 and GAP-3, Table 5 shows that |MVC| and $|MVC_\epsilon|$ are significantly smaller than Min(FA*, BA*), indicating that unidirectional search no longer is the best strategy for DXBB algorithms. This can be seen in the computational results as both MM_ϵ and GBFHS expand fewer nodes than Min(FA*, BA*), for both verifying optimality (Table 5) and for finding and verifying optimality (Table 2).

Examining the last column in Table 5 reveals that when GAP-0 is used, only about 26% of the total effort is needed to verify the optimality, whereas about 74% is needed to find the optimal path. This indicates that the optimal path was found relatively late during the search. For GAP-1, GAP-2, and GAP-3, the opposite occurs; most of the total effort is needed to verify optimality. This indicates that the optimal path is found relatively early during the search, but many more nodes must be expanded to verify the optimality of the path.

For the 15-puzzle, FA* and BA* required too much memory to be solved, hence the minimum vertex covers were not computed. Nevertheless, it is informative to execute the algorithms with an initial upper bound of C^* to determine how many nodes were expanded to verify the optimal path versus how many nodes were expanded to find and verify the optimal path. The computational results are presented in Table 6. The results for MM_ϵ and NBS are as indirectly reported in [5] and the results for $DVCBS_F$ and $DVCBS_A$ are as reported in [20]. Unlike Section 5.2, the results for GBFHS are from our implementation, because such experiments were not reported in [3]. Note that $DVCBS_A$ required fewer node expansions than $DVCBS_F$ to verify optimality, but required more expansions than $DVCBS_F$ to find the optimal path. DIBBS clearly dominates the other algorithms for verifying optimality. For unidirectional search, DIBBS only expanded 1.1% many nodes as IDA*. For bidirectional search, DIBBS only expanded 12.6% as many nodes as $DVCBS_A$, its closest competitor.

The computational results for the topspin problem are presented in Table 7. Similar to the results for the pancake problem, DIBBS expands fewer nodes, on average, than |MVC| and $|MVC_\epsilon|$ for the weaker heuristics ($M = 5, 6$ when $N = 16$ and $M = 7$ when $N = 18$). For the stronger heuristics, the average number of nodes expanded by DIBBS is within 13% of

Table 6
Minimum number of expansions for the 15-puzzle. The columns contain the average number of node expansions for the 100 instances.

Initial UB = C^*						
IDA*	MM_ϵ	NBS	GBFHS	$DVCBS_F$	$DVCBS_A$	DIBBS
120,885,331	13,000,000*	12,800,000*	12,438,228	11,589,837	10,659,744	**1,343,707**

* Chen et al. [5] reported that 99.4% of 13,162,312 MM_ϵ expansions had f-cost less than C^* and 99.7% of 12,851,889 NBS expansions had f-cost less than C^*.

Table 7
Minimum number of expansions for the topspin problem. The columns labeled FA*, BA*, Min(FA*, BA*), MM_ϵ, GBFHS, and DIBBS contain the average number of nodes expanded. If the entry is a percentage, then this is the percentage of the 100 instances that were successively solved before running out of memory.

Theoretical minimum number of expansions							Initial upper bound = C^*			
N	M	FA*	BA*	Min(FA*, BA*)	|MVC|	$\|MVC_\epsilon\|$	MM_ϵ	GBFHS	DIBBS	
									Expand	%
16	5	2,942,091	2,950,015	2,889,949	2,467,410	1,039,099	1,056,619	1,051,719	**985,245**	96%
16	6	332,840	335,424	324,148	324,148	324,148	527,198	526,023	**292,359**	94%
16	7	42,882	**42,318**	40,157	40,157	40,157	82,120	82,040	43,532	89%
16	8	**5,708**	5,762	5,378	5,378	5,378	11,432	11,434	6,085	63%
16	9	882	**879**	809	809	809	1,748	1,758	911	46%
18	7	2,723,514	2,678,516	2,602,543	2,602,543	2,602,543	4,744,282	4,701,183	**2,176,452**	95%
18	8	307,372	300,876	288,583	288,583	288,583	600,653	599,769	**297,685**	89%
18	9	38,260	**36,972**	35,087	35,087	35,087	75,059	75,041	39,217	73%

|MVC| for every combination of N and M. This compares quite favorably to MM_ϵ and GBFHS, which often expand twice as many nodes as |MVC|. Also similar to the results for the pancake problem, Table 7 shows that for the stronger heuristics, the number of nodes expanded by Min(FA*, BA*) is nearly identical to |MVC|. This indicates that unidirectional search dominates DXBB bidirectional search algorithms when the stronger heuristics are used.

7. Analysis of computational results

There are several ways that BHS algorithms can improve upon FA* or BA*. We want to investigate how these factors contributed to the overall success of DIBBS.

1. **One larger search replaced by two smaller searches.** To illustrate this point, suppose the graph has a uniform branching factor of b, all edges have unit cost, and heuristics are not used. Then the time complexity of Dijkstra's algorithm is $O\left(b^{C^*}\right)$, whereas the time complexity of Dijkstra's bidirectional algorithm is $O\left(b^{C^*/2}\right)$, because nodes with $g_d^*(v) > C^*/2$ are never expanded. Consequently, the bidirectional search is exponentially faster than the unidirectional search. In fact, this observation was the initial source of optimism that bidirectional heuristic search could outperform unidirectional heuristic search (A*) in a similar manner. As mentioned at the beginning of this paper, the anticipated improvement from two smaller searches, by and large, did not materialize in practice for earlier BHS algorithms. Furthermore, Kaindl and Kainz [12] proved that if a BHS algorithm uses $f_d(v)$ as the priority function and $UB \leq \max(f_{\min_f}, f_{\min_b})$ as the termination criteria, then the algorithm must expand either all the nodes with $f_f(v) < C^*$ or all the nodes with $f_b(v) < C^*$. Informally stated, such a BHS algorithm must expand at least as many nodes as FA* or BA*, ignoring nodes with $f_d(v) = C^*$.
   Part of the reason for the failure of BHS algorithms to benefit from two smaller searches is that most BHS algorithms expand nodes with $g_d^*(v) > C^*/2$. The meet in the middle (MM) family of algorithms [10,11,19] were specifically designed to prevent such nodes from being expanded by using $p_d(v) = \max(f_d(v), 2g_d(v))$ as the priority function and
   $$UB \leq \max\left(\min\left(p_{\min_f}, p_{\min_b}\right), f_{\min_f}, f_{\min_b}, g_{\min_f} + g_{\min_b} + \epsilon\right)$$
   as the termination criteria, where $p_{\min_d} = \min_{v \in O_d}\{p_d(v)\}$, $f_{\min_d} = \min_{v \in O_d}\{f_d(v)\}$, $g_{\min_d} = \min_{v \in O_d}\{g_d(v)\}$, and ϵ is the cost of the cheapest edge in the graph. By using a different priority function and a different termination criteria, MM is capable of solving problems while expanding fewer nodes than either FA* or BA*. Hence it has achieved the long sought goal of replacing one larger search by two smaller searches. Similarly, GBFHS achieves the goal by controlling where the two searches meet and implicitly using $g_{\min_f} + g_{\min_b} + \epsilon$ as part of the termination criteria.
   DIBBS also achieves the long sought goal of replacing one larger search by two smaller searches. For the pancake problem, Table 2 shows that the reduction in the number of nodes expanded by DIBBS compared to Min(FA*, BA*) is a factor of two for GAP-0 when $n = 40$, nearly a factor of three for GAP-1 when $n = 20$, and nearly a factor of 20 for

GAP-3 when $n = 10$. Furthermore, these reductions are not achieved merely by expanding fewer nodes with $f_d(v) = C^*$. In Table 5, C^* is supplied as the initial upper bound, so neither FA* nor BA* expand any nodes with $f_d(v) = C^*$, yet the reduction factor is nearly five for GAP-2 when $n = 10$ and nearly 14 for GAP-3 when $n = 10$. The results for the 15-puzzle are even more striking, where Table 3 shows that DIBBS reduces the number of node expansions by a factor of 115 compared to IDA*. Again, this reduction is not achieved merely by expanding fewer nodes with $f_d(v) = C^*$. Table 6 shows that when C^* is supplied as the initial upper bound, the reduction factor is 90. The results for the topspin problem are somewhat different. While Table 4 shows that DIBBS expands fewer nodes than Min(FA*, BA*), Table 7 indicates that this improvement may be primarily due to expanding fewer nodes with $f_d(v) = C^*$, i.e., by finding the optimal path sooner. Overall, the computational results demonstrate that DIBBS, like MM and GBFHS, is capable of solving problems while expanding fewer nodes than either FA* or BA*. The approach taken by MM and GBFHS is quite different than the approach taken by DIBBS. MM and GBFHS accomplish this goal by controlling where the two searches meet, whereas DIBBS does not control where the searches meet but uses dynamically improved bounds to reduce the size of the search. It is an area of future research to explore whether or not the two approaches can be successfully combined into a single algorithm.

The results discussed in the previous paragraph, especially for the extremely strong GAP-0 heuristic, convincingly demonstrate that DIBBS can overcome Barker and Korf's second conclusion that with a strong heuristic, BHS will expand more nodes than a unidirectional heuristic search. DIBBS achieves this by dynamically improving the bounds, or stated more generally, by using information from one search direction to aid the other direction. Their analysis implicitly assumed that such sharing of information would not happen.

2. **Exploit the easier direction.** For many combinatorial optimization problems, one search direction may be significantly easier than the other direction. For some such problems, the easier direction might always be the same. For example, Barker and Korf [2] reported that the forward direction is consistently easier than the backward direction for the peg solitaire problem. In such a case, one would choose to use FA* over BA*. But for many other problems, the better direction is dependent on the particular instance, in which case it is not known apriori whether it will be better to use FA* or BA*. Bidirectional search has the advantage that it can use information as the algorithm proceeds to determine which direction is easier and then place greater emphasis on that direction.

   For the pancake problem, Table 2 shows that the better direction is dependent on the instance when GAP-0 or GAP-1 is used. For example, for $n = 40$ with GAP-0, FA* expands 92,309 nodes on average, which is about 5% more nodes than BA*, which expands 88,234 nodes, on average. But if the better direction was known for each instance in advance, then the number of node expansions could be reduced to 56,702, on average, which represents a 35% reduction. While we do not know of a way to directly measure how much DIBBS benefited from exploiting the easier direction, it could not have not benefited more than 35%.

   Interestingly enough, when GAP-2 or GAP-3 is used, the backward direction is consistently better than the forward direction. This can be seen in Table 2 by observing that the number of nodes expanded by BA* and Min(FA*, BA*) are nearly identical, while the number of nodes expanded by FA* is much larger. For GAP-2 and GAP-3, the improvement of DIBBS over FA* may be partially due to exploiting the better direction, but the improvement over BA* is not.

3. **Find the optimal path sooner.** It is well known that for many combinatorial optimization problems there are a large number of nodes with $f(v) = C^*$. Both FA* and BA* do not need to expand any such nodes once the optimal path has been found. The pancake problem with the GAP-0 heuristic is a good example. For $n = 40$, Table 2 shows that when C^* is not known in advance, then FA* expanded 92,309 nodes on average. On the other hand, Table 5 shows that when C^* is supplied as the initial upper bound, then FA* only expanded 7,458 nodes on average. These are precisely the nodes with $f(v) < C^*$. Thus, FA* when C^* is not known in advance, expanded nearly 85,000 nodes with $f(v) = C^*$, on average. The results for BA* are very similar. FA* does not benefit from knowing C^* in advance when any of GAP-1, GAP-2, or GAP-3 is used. This happens because with these heuristics FA* finds the optimal path while expanding nodes with $f(v) = C^* - 1$. In contrast, BA* does benefit from knowing C^* in advance when any of GAP-1, GAP-2, or GAP-3 is used. Therefore, whether or not finding the optimal path sooner is helpful depends on the problem, the heuristic, and the direction.

   For the pancake problem, comparing Table 2 to Table 5 reveals that DIBBS does benefit from finding the optimal path sooner than FA* or BA*, especially for the strongest heuristic, GAP-0. For example, when GAP-0 is used for $n = 40$, then DIBBS expands about 72% fewer nodes than either FA* or BA*, but when C^* is supplied as the initial upper bound, then DIBBS only expands 13% (10%) fewer nodes than FA* (BA*). If this were the only data, it would appear to confirm Barker and Korf's conjecture that the possible contribution of Kaindl and Kainz's BHS algorithm was to avoid generating nodes with f cost equal to the cost of an optimal path. However, the data for the somewhat weaker heuristics, GAP-1, GAP-2, and GAP-3, prove otherwise. For example, when GAP-3 is used for $n = 10$, then DIBBS expands about 98% (95%) fewer nodes than FA* (BA*), but when C^* is supplied as the initial upper bound, then DIBBS expands 98% (92%) fewer nodes than FA* (BA*). In this case, finding the optimal path sooner is a relatively small component of the overall success of DIBBS over FA* and BA*.

   For the 15-puzzle, comparing Table 3 to Table 5 reveals that IDA* expands 115 times as many nodes as DIBBS when C^* is not known in advance and 90 times as many nodes when C^* is supplied as the initial upper bound. Hence, finding the optimal path sooner is a relatively small component of the overall success of DIBBS over IDA*. For the topspin problem, especially for the stronger heuristics, Tables 4 and 7 show that finding the optimal path sooner is a large

component of the overall success of DIBBS over FA* and BA*. For these problems, DIBBS actually requires slightly more node expansions to verify optimality than Min(FA*, BA*), but requires significantly fewer node expansions to find and verify optimality.

4. **Dynamically improved bounds.** Bidirectional search has the potential advantage of using information from one direction to improve the bounds in the backward direction. DIBBS uses dynamically improved bounds to avoid expanding nodes with $\min\left[\overline{f}_f^*(v), \overline{f}_b^*(v)\right] > C^*$ and to terminate the algorithm sooner.

It is difficult to measure how much each of the possible sources of improvement contribute to the overall success of DIBBS, or any other BHS algorithm, because they are intricately interwoven. For example, not only do dynamically improved bounds prevent the expansion of nodes with $\min\left[\overline{f}_f^*(v), \overline{f}_b^*(v)\right] > C^*$ and permit the search to be terminated earlier, but they also help to guide the search toward finding the optimal path sooner, which is beneficial in reducing the overall number of nodes expanded.

Declaration of competing interest

The authors 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 thank two anonymous reviewers for their thoughtful comments, resulting in a significantly improved manuscript. The second author was supported in part by the Air Force Office of Scientific Research under Grant No. FA9550-19-1-0106. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government, or the Air Force Office of Scientific Research.

References

[1] A. Auer, H. Kaindl, A case study of revisiting best-first vs. depth-first search, in: R.L. de Mantaras, L. Saitta (Eds.), Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'2004, Including Prestigious Applicants of Intelligent Systems, PAIS 2004, IOS Press, 2004, pp. 141–145.
[2] J.K. Barker, R.E. Korf, Limitations of front-to-end bidirectional heuristic search, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI-15, 2015, pp. 1086–1092.
[3] Mike Barley, Patricia Riddle, Carlos Linares López, Sean Dobson, GBFHS Ira Pohl, A generalized breadth-first heuristic search algorithm, in: Proceedings of the Eleventh International Symposium on Combinatorial Search, SoCS 2018, 2018, pp. 28–36.
[4] L. Bulteau, G. Fertin, I. Rusa, Pancake flipping is hard, J. Comput. Syst. Sci. 81 (8) (2015) 1556–1574.
[5] Jingwei Chen, Robert C. Holte, Sandra Zilles, Nathan Sturtevant, Front-to-end bidirectional heuristic search with near-optimal node expansions, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-2017, 2017, pp. 489–495.
[6] R. Dechter, J. Pearl, Generalized best-first search strategies and the optimality of A*, J. Assoc. Comput. Mach. 32 (3) (1985) 505–536.
[7] Jürgen Eckerle, Jingwei Chen, Nathan Sturtevant, Sandra Zilles, Robert C. Holte, Sufficient conditions for node expansion in bidirectional heuristic search, in: International Conference on Automated Planning and Scheduling, ICAPS-2017, 2017, pp. 79–87.
[8] Ariel Felner, Position paper: using early goal test in A*, in: Proceedings of the Eleventh International Symposium on Combinatorial Search, SoCS 2018, 2018, pp. 158–162.
[9] M. Helmert, Landmark heuristics for the pancake problem, in: A. Felner, N. Sturtevant (Eds.), Proceedings of the Third Annual Symposium on Combinatorial Search, SOCS-2010, AAAI Press, 2010, pp. 109–110.
[10] R.C. Holte, A. Felner, G. Sharon, N.R. Sturtevant, Bidirectional search that is guaranteed to meet in the middle, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI-16, 2016.
[11] R.C. Holte, A. Felner, G. Sharon, N.R. Sturtevant, J. Chen, MM: a bidirectional search that is guaranteed to meet in the middle, Artif. Intell. 252 (2017) 232–266.
[12] H. Kaindl, G. Kainz, Bidirectional heuristic search reconsidered, J. Artif. Intell. Res. 7 (1997) 283–317.
[13] R.E. Korf, Depth-first iterative-deepening: an optimal admissible tree search, Artif. Intell. 27 (1) (1985) 97–109.
[14] J.B. Kwa, BS*: an admissible bidirectional staged heuristic search algorithm, Artif. Intell. 38 (1) (1989) 95–109.
[15] I. Pohl, Bi-directional search, Mach. Intell. 6 (1971) 127–140.
[16] Samir K. Sadhukhan, Bidirectional heuristic search based on error estimate, CSI J. Comput. S1 (2013) 57.
[17] Eshed Shaham, Ariel Felner, Jingwei Chen, Nathan R. Sturtevant, The minimal set of states that must be expanded in a front-to-end bidirectional search, in: Proceedings of the Tenth International Symposium on Combinatorial Search, SoCS 2017, 2017, pp. 82–90.
[18] Eshed Shaham, Ariel Felner, Nathan R. Sturtevant, Jeffrey S. Rosenschein, Minimizing node expansions in bidirectional search with consistent heuristics, in: Proceedings of the Eleventh International Symposium on Combinatorial Search, SoCS 2018, 2018, pp. 81–89.
[19] G. Sharon, R.C. Holte, A. Felner, N.R. Sturtevant, Extended abstract: an improved priority function for bidirectional heuristic search, in: Proceedings of the Ninth International Symposium on Combinatorial Search, SoCS 2016, 2016, pp. 139–140.
[20] Shahaf S. Shperberg, Ariel Felner, Nathan R. Sturtevant, Solomon E. Shimony, Avi Hayoun, Enriching non-parametric bidirectional search algorithms, in: Proceeding of the Thirty-Third AAAI Conference on Artificial Intelligence, AAAI-19, 2019.
[21] Nathan R. Sturtevant, Ariel Felner, A brief history and recent achievements in bidirectional search, in: Proceeding of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI-18, 2018, pp. 8000–8008.
[22] Fan Yang, Joseph Culberson, Robert Holte, Uzi Zahavi, Ariel Felner, A general theory of additive state space abstractions, J. Artif. Intell. Res. 32 (2008) 631–662.