A Brief History and Recent Achievements in Bidirectional Search

Nathan R. Sturtevant
Computer Science Department
University of Denver
USA
sturtevant@cs.du.edu

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

Abstract

The state of the art in bidirectional search has changed significantly a very short time period; we now can answer questions about unidirectional and bidirectional search that until very recently we were unable to answer. This paper is designed to provide an accessible overview of the recent research in bidirectional search in the context of the broader efforts over the last 50 years. We give particular attention to new theoretical results and the algorithms they inspire for optimal and near-optimal node expansions when finding a shortest path.

Introduction and Overview

Shortest path algorithms have a long history dating to Dijkstra's algorithm (DA) (Dijkstra 1959). DA is the canonical example of a best-first search which prioritizes state expansions by their g-cost (distance from the start state). Historically, there were two enhancements to DA developed relatively quickly: bidirectional search and the use of heuristics.

Nicholson (1966) suggested *bidirectional search* where the search proceeds from both the start and the goal simultaneously. In a two dimensional search space a search to radius r will visit approximately r^2 states. A bidirectional search will perform two searches of approximately $(r/2)^2$ states, a reduction of a factor of two. In exponential state spaces the reduction is from b^d to $2b^{d/2}$, an exponential gain in both memory and time. This is illustrated in Figure 1, where the large circle represents a unidirectional search towards the goal, while the smaller circles represent the two parts of a bidirectional search.

Just two years later, DA was independently enhanced with admissible heuristics (distance estimates to the goal) that resulted in the A* algorithm (Hart, Nilsson, and Raphael 1968). A* is goal directed – the search is focused towards the goal by the heuristic. This significantly reduces the search effort required to find a path to the goal.

The obvious challenge was whether these two enhancements could be effectively combined into *bidirectional heuristic search* (Bi-HS). Pohl (1969) first addressed this challenge showing that in practice *unidirectional heuristic search* (Uni-HS) seemed to beat out Bi-HS. Many Bi-HS algorithms were developed over the years (see a short survey below), but no such algorithm was shown to consistently outperform Uni-HS.

Barker and Korf (2015) recently hypothesized that in most cases one should either use *bidirectional brute-force search* (Bi-BS) or Uni-HS (e.g. A*), but that Bi-HS is never the best approach. This work spurred further research into Bi-HS, and has lead to new theoretical understanding on the nature of Bi-HS as well as new Bi-HS algorithms (e.g., MM, fMM and NBS described below) with strong theoretical guarantees. The purpose of this paper is to provide a high-level picture of this new line of work while placing it in the larger context of previous work on bidirectional search.

While there are still many questions yet to answer, we have, for the first time, the full suite of analytic tools necessary to determine whether bidirectional search will be useful on a given problem instance. This is coupled with a Bi-HS algorithm that is guaranteed to expand no more than twice the minimum number of the necessary state expansions in practice. With these tools we can illustrate use-cases for bidirectional search and point to areas of future research.

Terminology and Background

We define a shortest-path problem as a n-tuple $(start, goal, exp_F, exp_B, h_F, h_B)$, where the goal is to find the least-cost path between $start$ and $goal$ in a graph G. G is not provided *a priori*, but is provided implicitly through the exp_F and exp_B functions that can expand and return the forward (backwards) successors of any state.

Bidirectional search algorithms interleave two separate searches, a search forward from $start$ and a search backward from $goal$. We use f_F, g_F and h_F to indicate f-, g-, and h-costs in the forward search and f_B, g_B and h_B similarly in the backward search. Likewise, $Open_F$ and $Open_B$ store states generated in the forward and backward directions, respectively. Finally, $gmin_F$, $gmin_B$, $fmin_F$ and $fmin_B$ denote the minimal g- and f-values in $Open_F$ and $Open_B$ respectively. $d(x, y)$ denotes the shortest distance between x and y.

Front-to-end algorithms use two heuristic functions. The *forward heuristic*, h_F, is *forward admissible* iff $h_F(u) \leq d(u, goal)$ for all u in G and is *forward consistent* iff $h_F(u) \leq d(u, u') + h_F(u')$ for all u and u' in G. The *backward heuristic*, h_B, is *backward admissible* iff $h_B(v) \leq$

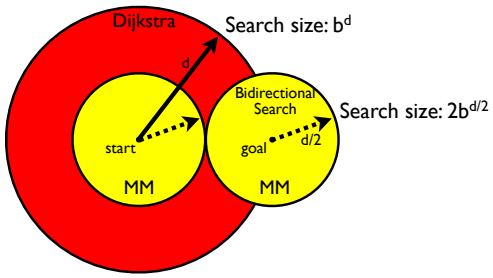

Figure 1: High-level illustration of unidirectional (large circle) and bidirectional (two small circles) search.

$d(start, v)$ for all v in G and is *backward consistent* iff $h_B(v) \leq d(v', v) + h_B(v')$ for all v and v' in G. $C^* = d(start, goal)$ is the cost of an optimal solution.

Our work assumes that search algorithms only have black-box access to the expand, heuristic, and cost functions. This follows the assumptions of Dechter and Pearl (1985) formalized in the description of *Deterministic, Expansion-based Black-box (DXBB)* algorithms (Eckerle et al. 2017).

Background

There are three main algorithmic design decisions on which previous Bi-HS algorithms differ: (1) the *stopping condition*, (2) the *selection of which state to expand next* and (3) the *nature of the heuristic*. We briefly cover these here; for more details see Holte et al. (2017).

Stopping condition A basic stopping condition (denoted the "$gmin$ stopping condition") refers to any stopping condition of the form $cost(U) \leq gmin_F + gmin_B$, where U the least-cost solution that the search has found so far found, and $cost(U)$ is the cost of that solution. This stopping condition is usually used for Bi-BS systems. These systems differ on whether this stopping condition is tested when generating or expanding states that appear in both frontiers (Nicholson 1966; Dreyfus 1967; Pohl 1969; 1971).

For Bi-HS algorithm a more informed stopping condition (denoted the "$fmin$ stopping condition") is to stop the search once $cost(U) \leq \max(fmin_F, fmin_B)$ Most previous Bi-HS algorithms used the $fmin$ stopping condition, the only difference being that some update U only when a state becomes closed in both directions (Barker and Korf 2012; Davis, Pollack, and Sudkamp 1984; Mohr and Pasche 1988; Pohl 1969) while others update U when a state becomes open in both directions (Hall 1971; Kowalski 1972; Kwa 1989; Pijls and Post 2010; Wilt and Ruml 2013). A unified stopping condition uses both of these and stops the search as soon as $cost(U) \leq \max\{gmin_F + gmin_B, fmin_F, fmin_B\}$. Other stopping conditions are covered in (Holte et al. 2017).

Selecting the state to expand For Bi-BS some methods strictly alternate between the search directions in a round robin fashion (Goldberg and Harrelson 2005; Helgason, Kennington, and Stewart 1993; Ikeda et al. 1994; Luby and Ragde 1989). Others select a state with the smallest g-value in either search direction (Nicholson 1966).

For Bi-HS, a simple selection policy chooses a state with the minimum f-value in either search direction but it was used relatively rarely (Pijls and Post 2010). A more common approach is the "cardinality criterion" (Pohl 1969). This approach chooses a search direction, such as the direction with the smaller open list, and then expands the state with the minimum f-value in the chosen direction. Many variations on this exist (Auer and Kaindl 2004; Kowalski 1972; Barker and Korf 2012). Others employ simple switching policies such as strict alternation between the two directions (Arefin and Saha 2010; Pijls and Post 2009; Sadhukhan 2012) or switching to maintain a fixed ratio between the number of states expanded in the two directions (Wilt and Ruml 2013). An extreme example is perimeter search (Dillenburg and Nelson 1994; Kaindl and Kainz 1997; Linares López and Junghanns 2002; Manzini 1995), which begins by doing a fixed amount of search in the backward direction and then does all the remaining search in the forward direction.

The heuristic used Most Bi-HS systems use *front-to-end* heuristics (Kaindl and Kainz 1997) that estimate the distance to $goal$ in the forward direction and to $start$ in the backward direction. By contrast, *front-to-front* algorithms use a heuristic $h(u, v)$ defined on any two states (u, v) and may perform heuristic lookups between any two states that belong to opposite frontiers (de Champeaux and Sint 1977; de Champeaux 1983; Davis, Pollack, and Sudkamp 1984; Politowski and Pohl 1984; Eckerle and Ottmann 1994; Arefin and Saha 2010).

Estimating paths between all pairs of states in the frontier of a search is expensive, and thus front-to-front searches are not usually practical. Single-frontier Bidirectional Search (SFBDS) (Felner et al. 2010; Lippi, Ernandes, and Felner 2012; 2016; Eckerle and Ottmann 1994) is the most practical implementation of this approach, but in this paper we only consider front-to-end algorithms.

Key Bidirectional Heuristic Search Insights

During the last 50 years, researchers have gained many insights into Bi-HS, attempting to explain why it has not been widely adopted. We briefly describe these insights next.

Frontiers Missing or Crossing One of the classic explanations was that the frontiers might miss each other (Nilsson 1980). This lead to techniques for pushing the search frontiers towards each other. Kwa, however, observed that the frontiers weren't missing, they were actually going through each other. As a result, BS* was developed with nipping and pruning techniques in order to prevent the search frontiers from crossing (Kwa 1989). While these techniques significantly improved the performance of bidirectional search, experimental results were still slightly worse than A*.

Time to Prove Optimality After further work, Kaindl and Kainz (1997) suggested that in addition to the frontiers crossing, there was another reason for the poor performance

of bidirectional search: the majority of effort in a bidirectional search is spent proving the optimal solution after the frontiers meet. This was the generally accepted understanding of bidirectional search for many years.

Only Bi-BS or Uni-HS Barker and Korf revived the discussion of bidirectional search with two conjectures (Barker and Korf 2015):

BK1: Uni-HS will expand fewer states than Bi-HS if more than half of the states expanded by Uni-HS have $g \leq C^*/2$, where C^* is the optimal solution cost.

BK2: If fewer than half of the states expanded by Uni-HS using heuristic h have $g \leq C^*/2$, then adding h to Bi-BS will not decrease the number of states it expands.

These conjectures were made based on a number of assumptions: that the forward and backwards search were symmetric, and that the forward and backwards searches met in the middle – that in each direction the search will not expand a state with $g > C^*/2$, a property that no bidirectional search algorithm had at the time.

Meet in the Middle (MM) This inspired the development of the MM algorithm (Holte et al. 2016; 2017). MM is the first bidirectional heuristic search algorithm guaranteed to "meet in the middle". That is, MM will never expand a state whose g-value exceeds $C^*/2$. That is, the bidirectional picture in Figure 1 is accurate for MM in that the states expanded by MM will not go outside these circles, and the searches will meet at the middle of the optimal solution. The picture is not precise for an algorithm like BS* because the frontiers are not constrained to stay within the circles.

This is achieved by MM's novel selection criterion and priority functions, $pr_F(u)$ and $pr_B(u)$ for the forward and backwards directions, respectively:

$$pr_F(u) = \max(g_F(u) + h_F(u), 2 \cdot g_F(u))$$
$$pr_B(u) = \max(g_B(u) + h_B(u), 2 \cdot g_B(u))$$

MM expands a state with minimum priority from either direction. It uses a novel stopping condition that combines the $gmin$ and the $fmin$ stopping condition functions.

We provide a brief argument of why MM meets in the middle here. (We present a forward argument, the backward argument is analogous.) (1) Any state s_f on the optimal path with $g_F(s_f) \leq \frac{1}{2}C^*$ has $pr_F(s_f) \leq C^*$. This follows from the admissibility of the heuristic. (2) All states t_f with $g_f(t_f) > \frac{1}{2}C^*$ have $pr_F(t_f) > C^*$. This follows because $2g_f(t_f) > C^*$. Since states are expanded from low priority to high priority, the optimal path up to the middle of the search is guaranteed to be explored in both directions before any states with $g_f(t_f) > \frac{1}{2}C^*$. Since expanding the optimal path will terminate the search, no state t_f will be expanded.

Fractional MM While MM meets exactly in the middle, Shaham et al. (2017) showed that the approach can be generalized to allow the search to meet at any fraction of the optimal solution cost. They proposed a generalization of MM called *Fractional MM* (fMM(p)) which uses the following priority functions on paths in the open lists:

$$pr_F(u) = \max(g_F(u) + h_F(u), g_F(u)/p)$$
$$pr_B(u) = \max(g_B(u) + h_B(u), g_B(u)/(1-p))$$

where $0 < p < 1$.

fMM(p)'s forward and backward searches meet at $(pC^*, (1-p)C^*)$ by similar reasoning for why MM meets in the middle. MM is a special case of fMM(p) for $p = 1/2$. Forward A* and reverse A* (searching from $goal$ to $start$) are also special cases corresponding to $p = 1$ and $p = 0$ (although division by zero must be avoided).

Other Variants of MM Other variants of MM include MMϵ (Sharon et al. 2016) which has a slightly improved priority function, and PEMM (Sturtevant and Chen 2016) which uses parallel and external memory search to scale the size of problems that can be solved. For instance, PEMM is able to solve Rubik's cube instances at maximal depth (20).

Bidirectional Search in Planning One place that bidirectional search has found traction is in the area of planning. Bidirectional planners have been entered into the International Planning Competition, and in the 2014 International Planning Competition (Vallati et al. 2015), the best Sequential Optimal planner was the SymBA* planner (Torralba et al. 2014). This planner uses symbolic search with dynamically-built heuristics.

Bi-HS Anomaly If h_1 and h_2 are admissible heuristics and $h_1(s) > h_2(s)$ for all non-goal states, then A* cannot expand more distinct states with h_1 than with h_2 (Nilsson 1980). In particular, A* with a non-zero heuristic cannot expand more states than Uni-BS. In bidirectional search, however, this property does not hold (Holte et al. 2017).

Optimal and Near-Optimal Bi-HS

In this section we summarize three recent papers that together provide a clear picture of bidirectional search. The first paper looked at the states which must necessarily be expanded when solving a bidirectional search problem (Eckerle et al. 2017). The second paper showed that fMM for some value of p (which must be provided by an oracle or found offline) expands exactly the minimum number of state expansions that are needed to solve a problem (Shaham et al. 2017). The third paper developed NBS, which can find the optimal solution while expanding no more than two times the minimal number of necessary states (Chen et al. 2017).

Sufficient Conditions for Uni-HS

An important historical question for A* was whether A* was the best possible algorithm, or whether there could be a better one. Dechter and Pearl showed that, with a consistent heuristic, A* is optimal. In particular, any state with $f_F < C^*$ must be expanded (Dechter and Pearl 1985). We illustrate this in the top of Figure 2. If an algorithm does not expand some state u with $f_F(u) < C^*$, we could create a new problem instance by adding an edge from u to the $goal$ on which the algorithm would not find the optimal path. Thus, knowing that $f_F(u) < C^*$ is sufficient to prove that an optimal Uni-HS algorithm will expand u.

Sufficient Conditions for Bidirectional Search

The picture in bidirectional search is more complicated. We illustrate this in the bottom of Figure 2. Here, the path to

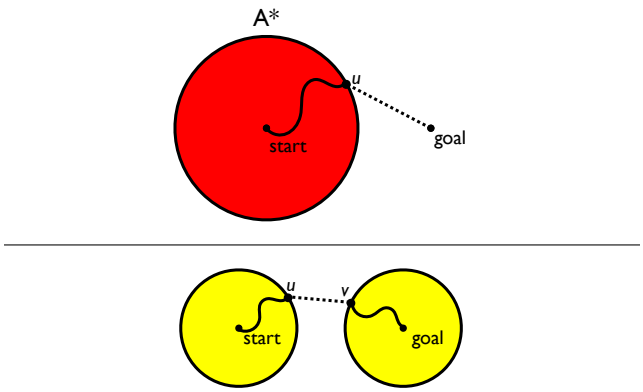

Figure 2: A* necessary expansions (top) vs. Bi-HS necessary expansions (bottom)

the goal goes from $start$ to u to v to the $goal$. Thus, the sufficient conditions must reason about both u and v to see if there might be a path between them.

Eckerle et al. (2017) developed three conditions for state expansion in bidirectional search that reason about potential paths between u and v. If all conditions are met, the search must explore to see if there is a shorter path between u and v. The first two arise directly from the conditions for unidirectional search, since the cost from u to v to the $goal$ is lower-bounded by the estimate from u to the $goal$ directly. The third condition is the result of using the g-cost of a state in the forward (backward) direction as a heuristic for the backward (forward) search. Thus, the three conditions are:

$$f_F(u) < C^* \quad (1)$$

$$f_B(v) < C^* \quad (2)$$

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

If all three conditions are met and we do not explore between u and v, then there might be a different problem instance (with smaller C^*) on which the optimal solution is missed. Thus, in order to verify whether or not that path exists we either have to expand u or v. Either expansion suffices to find the next step on the path or prove that it does not exist. We do not, however, have to expand both states.

These conditions can be stated concisely as follows.

Definition 1. *For each pair of states (u, v) let*

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

If a pair of states (u, v) has $lb(u, v) < C^*$ then it is sufficient to expand one of u or v to find the optimal solution. As a result, we call (u, v) a *must-expand* pair, meaning that one of the pair is *necessarily* expanded by all optimal algorithms.

Note that this theory does not discuss what to do with state pairs that have $lb(u, v) = C^*$. In this theory such expansions are not *necessary*, although we can construct problems with many such states. In practice these states do not make a significant contribution to the total number of state expansions, so we focus only on necessary expansions.

The *Must-Expand Graph* G_{MX}

The unique property of the sufficient condition is that the condition contains an *or*. Such a condition is also found in the vertex cover problem. So, we will represent the sufficient conditions as a vertex cover on a graph G_{MX}.

Definition 2. *The Must-Expand Graph G_{MX} of problem instance is an undirected, unweighed bipartite graph defined as follows. For each state $u \in G$, there are two vertices in G_{MX}, the left vertex u_F and right vertex u_B. For each pair of states $u, v \in G$, there is an edge in G_{MX} between u_F and v_B if and only if $lb(u_F, v_B) < C^*$. Thus, there is an edge in G_{MX} between u_F and v_B if and only if the pair (u, v) is a must-expand pair.*

We illustrate G_{MX} in Figure 3. In part (a) we show a graph with each state labeled with its forward and backwards h-cost. Part (b) of the figure shows the corresponding G_{MX} graph. Given an optimal solution cost of 3 there is an edge between c_F and e_B because the sum of their g-costs is less than 3 and the f-costs of c_F and e_B are also less than 3. The edges in G_{MX} exactly correspond to the state pairs that must be expanded, and therefore any vertex cover for G_{MX} will, by definition, represent a set of expansions that covers all the required state pairs (Chen et al. 2017).

In our example, one possible vertex cover includes the vertices in the left side with at least one edge in G_{MX}: $\{a_F, c_F, d_F, e_F\}$. This represents expanding all the required state pairs in the forward direction. This requires four expansions and is not optimal because the required state pairs can be covered with just three expansions: a in the forward direction and f and e in the backward direction. This corresponds to a minimal vertex cover of G_{MX}: $\{a_F, f_B, e_B\}$.

If we wish to know the minimum number of state expansions necessary for solving a bidirectional search problem, we must find the size of the minimum vertex cover of G_{MX}. We use VC to refer to the minimum vertex cover of G_{MX} and $|\text{VC}|$ to refer to the size of the minimum vertex cover.

Minimal Vertex Cover of G_{MX}

A key question about the VC is whether there exists a search algorithm that can expand the VC in practice. For instance, there is no search algorithm that can expand just $\{e_F, c_B\}$ in Figure 3, because any search algorithm must start by expanding the start or the goal. So, if this was the minimum vertex cover, it would not be achievable in practice. However, it ends up that the structure of G_{MX} is such that we can compute VC efficiently, and there always exist a bidirectional algorithm (fMM for some p) that will expand exactly the nodes of VC in practice (among nodes that are part of must expanded pairs) (Shaham et al. 2017).

In particular, if state s with forward g-cost $g_F(s)$ is in VC then every state t with $g_F(t) \leq g_F(s)$ will also be in VC, and analogously for the backward direction. This means that any VC contains a contiguously set of nodes from the start/goal in each direction. It is simpler to visualize an abstraction of G_{MX} which we denote by WG_{MX}. In WG_{MX} all k states with the same g-cost are abstracted together to form a single node with weight k. The *weighted vertex cover* of WG_{MX} is identical to VC. We show this

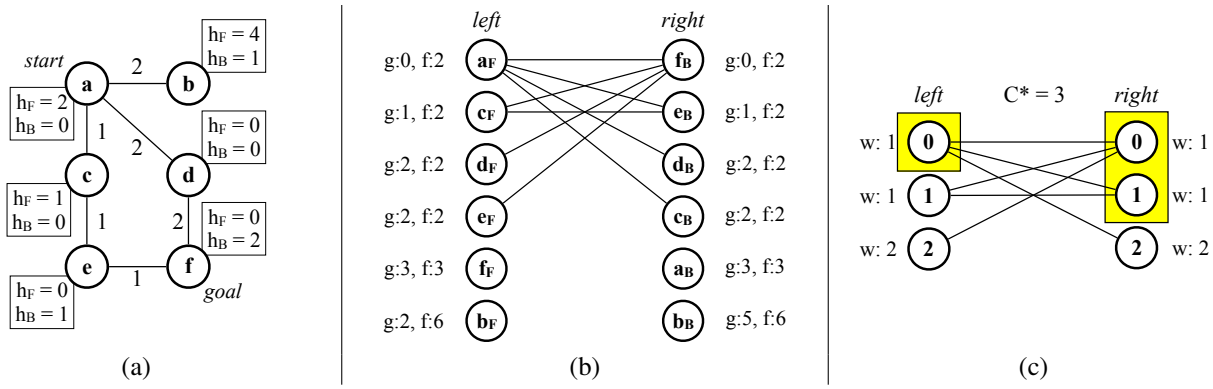

Figure 3: (a) A sample graph (b) the associated G_{MX} and (c) the weighted G_{MX} with the minimum vertex cover.

in Figure 3(c), where each state is labeled inside with its g-cost, and on the outside with its weight. (The state labels are not drawn, but they can be inferred.) One possible minimum vertex cover is highlighted on each side. The class of vertex covers which contain a contiguous set of g-costs in each direction are called *restrained*.

Definition 3. *Assume start and goal states for which $d(start, goal) = C^*$. Let S_F be a set of states in the forward direction and let S_B be a set of states in the backward direction. We say that the pair of sets (S_F, S_B) is* **strongly restrained** *if there exists a fraction $0 \leq p \leq 1$ such that S_F contains all states u with $g_F(u) < pC^*$ and no states with $g_F(u) \geq pC^*$ and S_B contains all states v with $g_B(v) < (1-p)C^*$ and no states with $g_B(v) \geq (1-p)C^*$.*

This condition resembles the priorities used by fMM. In fact, fMM(p) is strongly restrained with respect to p. Furthermore, there exists a fraction $0 \leq p^* \leq 1$ such that fMM(p^*) is *optimally effective* (Shaham et al. 2017), i.e., expands the necessary and sufficient must-expanded pairs. This fraction determines what states are included (and not included) in VC from each half of the vertex cover. The example in Figure 3 has two possible values of p^*, $\frac{1}{3}$ and $\frac{2}{3}$.

The main importance of fMM is theoretical. The exact value for p^* is not known in advance as it is a function of the given problem instance, G_{MX}, and C^*. Therefore, fMM(p^*) is not a DXBB algorithm because p^* is not known in advance. Furthermore, Chen et al. (2017) showed that for every DXBB algorithm there exists a worst-case graph where it will expand at least twice as many nodes as are in VC. Therefore, unlike unidirectional search, no DXBB bidirectional search algorithm can be guaranteed to be optimal in terms of state expansions.

However, given C^* and G_{MX}, there is a simple algorithm that can find p^* in time linear in the size of G_{MX}. Finding p^* and then running fMM(p^*) can serve for research purposes as an oracle – any algorithm can now be compared to the theoretical optimal algorithm, as we do below in our experimental section.

Algorithm 1 NBS

```
1:  Put start and goal onto Open_F and Open_B
2:  while Open_F and Open_B both not empty do
3:      Among u ∈ Open_F and v ∈ Open_B
4:      Select the pair (u, v) with lowest lb(u, v)
5:      if lb(u, v) ≥ cost(U) then
6:          return U
7:      end if
8:      Expand both u and v
9:      if New path from start to goal is found then
10:         Update U if new path is better than existing path
11:     end if
12: end while
```

Near-Optimal Bidirectional Search

We conclude by describing Near-Optimal Bidirectional Search (NBS) (Chen et al. 2017). NBS efficiently finds a near-optimal vertex cover of G_{MX} and thus its necessary state expansions are near-optimal. It is inspired by a greedy algorithm for finding a vertex cover (Papadimitriou and Steiglitz 1982) that selects an edge with two uncovered adjacent vertices, and then adds both vertices to the vertex cover.

NBS adapts the greedy algorithm for heuristic search problems; high-level pseudo-code is found in Algorithm 1. The best solution found is stored in U. NBS it chooses a pair of states (u, v) from G_{MX} with minimal $lb(u, v)$ (line 4). While only one of these states must be included in VC, NBS expands both of them (line 8). NBS requires careful implementation of data structures to ensure that the selection of (u, v) can be done correctly and efficiently.

NBS has the following properties. (1) It is guaranteed to find the optimal solution. (2) It expands at most $2|\text{VC}|$ states. So, the number of necessary state expansions will be no more than two times the minimal number of necessary state expansions. (3) No DXBB algorithm can have better worst-case performance. Taken together, these properties make it a powerful general purpose search algorithm, particularly because A* has no worst-case performance guarantee when compared to bidirectional search algorithms.

Table 1: Comparing the necessary state expansions on three domains with different quality of heuristics.

Domain	Heuristic Qual.	Forward A*	Backward A*	Bi-BS	NBS	Min Uni-HS˙	Offline fMM
Pancake	Strong	**153,518**	156,323	–	278,402	143,022	143,022
TOH4	Strong	19,339,936	**5,607,372**	6,670,903	6,283,143	5,607,372	5,154,787
Roads (CO-time)	Medium	115,200	123,152	112,946	**89,048**	83,466	70,635
Grid Mazes	Weak	57,427	56,993	25,469	**25,087**	41,953	23,795

Experiments

Other recent papers contain extensive experimental results (Barker and Korf 2015; Holte et al. 2017; Sturtevant and Chen 2016; Chen et al. 2017) illustrating bidirectional and unidirectional search. So, our focus here is on illustrating performance using the Forward A*, Backward A*, Bi-BS and NBS algorithms. We additionally provide the results of two oracles. The first oracle (Min Uni-HS) performs a unidirectional A* search, but always searches in the direction (forward or backward) that minimizes node expansions. The second oracle provides p^* to fMM and computes the minimum possible number of node expansions.

We analyze four domains in turn, beginning with the pancake puzzle. We experiment with 100 hard instances (Valenzano and Yang 2017) of the 20-pancake puzzle using the GAP heuristic (Helmert 2010). Table 1 contains average necessary state expansions, although Bi-BS runs out of memory when solving these problem instances. We see that there is only a small difference between forward and backwards search, and NBS is significantly worse than unidirectional search. The oracles reveal that on these problems either forward or backwards A* is always optimal, since both oracles perform the same number of average state expansions. NBS does almost exactly two times the minimum number of necessary state expansions. The use of these oracles shows no benefit to bidirectional search on this domain.

Our second domains is the 4-peg Towers of Hanoi (TOH4) with 14 disks. Our heuristic is the sum of a 10-disk and 4-disk PDB, and results are averaged over 50 problems. The start state is randomly generated and the goal is the canonical goal state. On these problems we see that backwards search is much cheaper than forward search. This is due to the smaller branching factor at the goal. While NBS does slightly worse than backwards A*, our oracle reveals that bidirectional search could be about 10% better than unidirectional search given the right search parameters. A more important insight in this domain is that if we accidentally search the wrong direction or if the start and goal were randomized, A* would have poor performance (the average of the forward and backwards search). But, these changes would have no impact on the performance of NBS.

Our third domain tests 200 random problems from the road network of Colorado, where edge costs are based on travel time, and the heuristic is Euclidean distance divided by the maximum travel speed. The Barker and Korf analysis does not apply to this domain because it is not symmetric. In this domain NBS does about 30% better than unidirectional search and Bi-BS. Our first oracle reveals that on an instance-by-instance basis there is a significant difference between unidirectional forward and backwards search, because of the minimum of the two is even better than bidirectional search. But, the bidirectional oracle shows that there is still further room for improvement for bidirectional search.

Our final experiment is in grid-based mazes of size 512x512 with corridors width one taken from the MovingAI benchmark repository (Sturtevant 2012). There are 12,000 problem instances of increasing difficulty; we use the octile-distance heuristic for the search. Here, unidirectional search is always worse than bidirectional search, even with an oracle. Furthermore, the heuristic does not provide much benefit to NBS, only reducing the average state expansions by 382 (less than a 2% reduction). In practice Bi-BSis faster because it does not have to lookup a heuristic value at each state. The fMM oracle show that there isn't much additional gain that could be achieved by a better bidirectional search algorithm in this domain; the problem is a weak heuristic.

Conclusions and Future Directions

The landscape of bidirectional search has changed significantly in the past few years. We now know that minimizing node expansions in bidirectional search is equivalent to solving a weighted minimum vertex cover and that VC is strongly restrained. Furthermore, we have a general-purpose algorithm, NBS, that will be within a factor of two of the minimum number of node expansions by any DXBB front-to-end search algorithm.

Given this foundation, there are many questions that still need to be addressed. These include question of suboptimal search, front-to-front search, and a better characterization of how our problems and heuristics influence the success of bidirectional search.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1551406. The work was sponsored by Israel Science Foundation (ISF) under grant #844/17.

References

Arefin, K. S., and Saha, A. K. 2010. A new approach of iterative deepening bi-directional heuristic front-to-front algorithm (IDBHFFA). *International Journal of Electrical and Computer Sciences (IJECS-IJENS)* 10(2).

Auer, A., and Kaindl, H. 2004. A case study of revisiting best-first vs. depth-first search. In *ECAI*, 141–145.

Barker, J. K., and Korf, R. E. 2012. Solving peg solitaire with bidirectional BFIDA*. In *AAAI*, 420–426.

Barker, J. K., and Korf, R. E. 2015. Limitations of front-to-end bidirectional heuristic search. In *AAAI*, 1086–1092.

Chen, J.; Holte, R. C.; Zilles, S.; and Sturtevant, N. R. 2017. Front-to-end bidirectional heuristic search with near-optimal node expansions. In *IJCAI. Also avaialble at http://arxiv.org/abs/1703.03868*.

Davis, H. W.; Pollack, R. B.; and Sudkamp, T. 1984. Towards a better understanding of bidirectional search. In *AAAI*, 68–72.

de Champeaux, D., and Sint, L. 1977. An improved bidirectional heuristic search algorithm. *J. ACM* 24(2):177–191.

de Champeaux, D. 1983. Bidirectional heuristic search again. *J. ACM* 30(1):22–32.

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

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1(1):269–271.

Dillenburg, J. F., and Nelson, P. C. 1994. Perimeter search. *Artificial Intelligence* 65(1):165–178.

Dreyfus, S. E. 1967. An appraisal of some shortest path algorithms. Technical Report RM-5433-PR, Rand Corporation.

Eckerle, J., and Ottmann, T. 1994. An efficient data structure for bidirectional heuristic search. In *ECAI*, 600–604.

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

Felner, A.; Moldenhauer, C.; Sturtevant, N. R.; and Schaeffer, J. 2010. Single-frontier bidirectional search. In *AAAI*.

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

Hall, P. A. V. 1971. Branch-and-bound and beyond. In *IJCAI*, 641–650.

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

Helgason, R. V.; Kennington, J. L.; and Stewart, B. D. 1993. The one-to-one shortest-path problem: An empirical analysis with the two-tree Dijkstra algorithm. *Computational Optimization and Applications* 2(1):47–75.

Helmert, M. 2010. Landmark heuristics for the pancake problem. In *Symposium on Combinatorial Search*, 109–110.

Holte, R. C.; Felner, A.; Sharon, G.; and Sturtevant, N. R. 2016. Bidirectional search that is guaranteed to meet in the middle. In *AAAI Conference on Artificial Intelligence*, 3411–3417.

Holte, R. C.; Felner, A.; Sharon, G.; Sturtevant, N. R.; and Chen, J. 2017. Bidirectional search that is guaranteed to meet in the middle. *Artificial Intelligence Journal (AIJ), In press*.

Ikeda, T.; Hsu, M.-Y.; Imai, H.; Nishimura, S.; Shimoura, H.; Hashimoto, T.; Tenmoku, K.; and Mitoh, K. 1994. A fast algorithm for finding better routes by AI search techniques. In *Proc. Vehicle Navigation and Information Systems Conference*, 291–296.

Kaindl, H., and Kainz, G. 1997. Bidirectional heuristic search reconsidered. *JAIR* 7:283–317.

Kowalski, R. A. 1972. AND/OR graphs, theorem-proving graphs, and bidirectional search. In Meltzer, B., and Michie, D., eds., *Machine Intelligence*, volume 7. Edinburgh Univ. Press. 167–194.

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

Linares López, C., and Junghanns, A. 2002. Perimeter search performance. In *Proc. 3rd International Conference on Computers and Games (CG)*, 345–359.

Lippi, M.; Ernandes, M.; and Felner, A. 2012. Efficient single frontier bidirectional search. In *Symposium on Combinatorial Search*.

Lippi, M.; Ernandes, M.; and Felner, A. 2016. Optimally solving permutation sorting problems with efficient partial expansion bidirectional heuristic search. *AI Commun.* 29(4):513–536.

Luby, M., and Ragde, P. 1989. A bidirectional shortest-path algorithm with good average-case behavior. *Algorithmica* 4(1):551–567.

Manzini, G. 1995. BIDA: an improved perimeter search algorithm. *Artificial Intelligence* 75(2):347–360.

Mohr, T., and Pasche, C. 1988. A parallel shortest path algorithm. *Computing* 40(4):281–292.

Nicholson, T. A. J. 1966. Finding the shortest route between two points in a network. *The Computer Journal* 9(3):275–280.

Nilsson, N. J. 1980. *Principles of Artificial Intelligence*. Springer.

Papadimitriou, C. H., and Steiglitz, K. 1982. *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

Pijls, W., and Post, H. 2009. A new bidirectional algorithm for shortest paths. *European Journal of Operational Research* 198:363–369.

Pijls, W., and Post, H. 2010. Note on "A new bidirectional algorithm for shortest paths". *European Journal of Operational Research* 207(2):1140–1141.

Pohl, I. 1969. Bi-directional and heuristic search in path problems. Technical Report 104, Stanford Linear Accelerator Center.

Pohl, I. 1971. Bi-directional search. *Machine Intelligence* 6:127–140.

Politowski, G., and Pohl, I. 1984. D-node retargeting in bidirectional heuristic search. In *Proc. National Conference on Artificial Intelligence (AAAI)*, 274–277.

Sadhukhan, S. K. 2012. A new approach to bidirectional heuristic search using error functions. In *Proc. 1st International Conference on Intelligent Infrastructure at the 47th Annual National Convention Comptuer Society of India (CSI-2012)*.

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

Sharon, G.; Holte, R. C.; Felner, A.; and Sturtevant, N. R. 2016. Extended abstract: An improved priority function for bidirectional heuristic search. In *SoCS*, 139–140.

Sturtevant, N., and Chen, J. 2016. External memory bidirectional search. In *IJCAI*, 676–682.

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

Torralba, A.; Alcazar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. SymBA*: A symbolic bidirectional A* planner. In *International Planning Competition*, 105–108.

Valenzano, R. A., and Yang, D. S. 2017. An analysis and enhancement of the gap heuristic for the pancake puzzle. In *Symposium on Combinatorial Search*, 109–118.

Vallati, M.; Chrpa, L.; Grzes, M.; McCluskey, T.; Roberts, M.; and Sanner, S. 2015. The 2014 international planning competition: Progress and trends. *AI Magazine*.

Wilt, C. M., and Ruml, W. 2013. Robust bidirectional search via heuristic improvement. In *AAAI*, 954–961.