Bidirectional Search That Is Guaranteed to Meet in the Middle

Robert C. Holte
Computing Science Department
University of Alberta
Edmonton, Canada T6G 2E8
(rholte@ualberta.ca)

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

Guni Sharon
ISE Department
Ben-Gurion University
Be'er-Sheva, Israel
(gunisharon@gmail.com)

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

Abstract

We present MM, the first bidirectional heuristic search algorithm whose forward and backward searches are guaranteed to "meet in the middle", i.e. never expand a node beyond the solution midpoint. We also present a novel framework for comparing MM, A*, and brute-force search, and identify conditions favoring each algorithm. Finally, we present experimental results that support our theoretical analysis.

1 Introduction and overview

Bidirectional search algorithms interleave two separate searches, a normal search forward from the start state, and a search backward (i.e. using reverse operators) from the goal.

Barker and Korf (2015)'s comparison of unidirectional heuristic search (Uni-HS, e.g. A*), bidirectional heuristic search (Bi-HS), and bidirectional brute-force search (Bi-BS) has two main conclusions (for caveats, see their Section 3):

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

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

A central assumption made by Barker and Korf is that the forward and backward searches comprising the bidirectional search never expand a node whose g-value (in the given direction) exceeds $C^*/2$. We say that a bidirectional search "meets in the middle" if it has this property.

This assumption raises a difficulty in applying their theory, because no known Bi-HS algorithm is guaranteed to meet in the middle under all circumstances. Papers on "front-to-front"[1] Bi-HS typically claim their searches meet in the middle, but none of them has a theorem to this effect (Arefin and Saha 2010; de Champeaux 1983; de Champeaux and Sint 1977; Davis, Pollack, and Sudkamp 1984; Eckerle 1994; Politowski and Pohl 1984). "Front-to-end" Bi-HS algorithms (Auer and Kaindl 2004; Ikeda et al. 1994; Kaindl and Khorsand 1994; Kwa 1989; Pohl 1969; Sadhukhan 2012) are not specifically designed to meet in the middle, and often they do not. For example, in Barker and Korf's Towers of Hanoi experiment BS* (Kwa 1989) often expanded nodes at depth 13 in each direction even though the solution lengths C^* were at most 16.

To remedy this we present a new front-to-end Bi-HS algorithm, MM, that is guaranteed to meet in the middle. MM_0 is the brute-force ($h(s) = 0\ \forall s$) version of MM. We also present a new framework for comparing MM_0, unidirectional brute-force search (Uni-BS), MM, and A* that allows a precise characterization of the regions of the state space that will be expanded by one method but not another. We use this to identify conditions under which one method will expand fewer nodes than another, and conditions guaranteeing BK1's correctness. We also show that, unlike unidirectional search, adding a non-zero heuristic ($\neq 0$ for every non-goal node) to Bi-BS can cause it to expand more nodes. For example, MM expands 4 times more nodes than MM_0 in one of our Pancake Puzzle experiments. Overall, our experiments on the Pancake Puzzle and Rubik's Cube show that the algorithm expanding the fewest nodes could be any one of Uni-HS, MM_0 or MM, depending on the heuristic used.

Although we introduce a new algorithm (MM), we do not present an experimental comparison of MM to existing Bi-HS algorithms, and we do not claim that MM_0 and MM are the best bidirectional search algorithms in terms of minimizing run time or the number of nodes expanded. These issues are outside the scope of this paper. Like the Barker and Korf paper, this paper is theoretical. MM's significance is that it is the only Bi-HS algorithm to which our analysis, and Barker and Korf's, applies. These theories give strong justification for bidirectional search algorithms that meet in the middle. As the first of its breed, MM represents a new direction for developing highly competitive Bi-HS algorithms. A thorough

[1] In bidirectional heuristic search, there are two different ways to define the heuristic function (Kaindl and Kainz 1997). A "front-to-end" heuristic—$h_F(n)$ for the forward search and $h_B(n)$ for the backward search—directly estimates the distance from node n to the target of the search (the target for the forward search is the goal, the target for the backward search is the start state). By contrast a "front-to-front" heuristic estimates the distance from n to the search target indirectly. For forward search it is defined as $h_F(n) = \min_{m \in Open_B}\{h(n,m) + g_B(m)\}$ where $Open_B$ is the backward search's open list, $h(n,m)$ is a function estimating the distance between any two nodes, and $g_B(m)$ is the g-value of node m in the backward search. A front-to-front heuristic for the backward search is defined analogously.

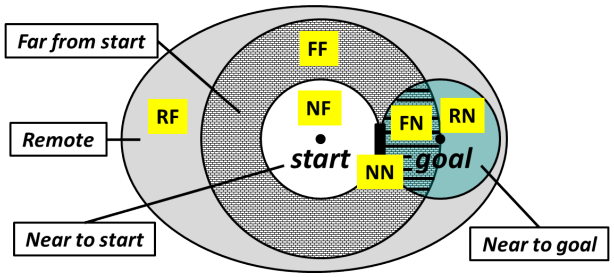

Figure 1: Diagrammatic depiction of the different regions.

empirical evaluation of MM is an important study that we will undertake in the future.

2 Definitions and terminology

A problem instance is a pair $(start, goal)$ of states in a state-space in which all edge weights are non-negative. The aim of search is to find a *least-cost path* from $start$ to $goal$. $d(u, v)$ is the distance (cost of a least-cost path) from state u to state v. $C^* = d(start, goal)$ is the cost of an optimal solution.

We use the usual notation—$f, g, Open$, etc.—and use $gmin$ and $fmin$ for the minimum g- and f-value on $Open$. We have separate copies of these variables for the two search directions, with a subscript (F or B) indicating the direction:
Forward search: f_F, g_F, h_F, $Open_F$, $Closed_F$, etc.
Backward search: f_B, g_B, h_B, $Open_B$, $Closed_B$, etc.

We assume that each search direction uses an admissible front-to-end (Kaindl and Kainz 1997) heuristic.

We say state s is "near to $goal$" if $d(s, goal) \leq C^*/2$, and "far from $goal$" otherwise. For $start$, we make a 3-way distinction: s is "near to $start$" if $d(start, s) \leq C^*/2$, "far from $start$" if $C^*/2 < d(start, s) \leq C^*$, and "remote" if $d(start, s) > C^*$. These categories divide the state-space into 6 disjoint regions shown in Figure 1. We denote these regions by two letter acronyms. The first letter indicates the distance from $start$ (N=near, F=far, R=remote) and the second letter indicates the distance from $goal$ (N=near, F=far). For example, FN is the set of states that are far from $start$ and near to $goal$. NN includes only those states at the exact midpoint of optimal solutions. None of the search algorithms in this paper expands a state in RF.

In Sections 4–7 we compare MM_0, MM, Uni-BS, and A* based mainly on the nodes they expand in each region. A region's name denotes both the set of states and the number of states in the region. We will use the names in equations and inequalities. An inequality involving two algorithms, e.g. A* < MM, indicates that one algorithm (A* in this example) expands fewer nodes than the other.

3 MM: A Novel Bi-HS Algorithm

MM runs an A*-like search in both directions, except that MM orders nodes on the Open list in a novel way. The priority of node n on $Open_F$, $pr_F(n)$, is defined to be:

$$pr_F(n) = \max(f_F(n), 2g_F(n)).$$

$pr_B(n)$ is defined analogously. We use $prmin_F$ and $prmin_B$ for the minimum priority on $Open_F$ and $Open_B$,

Algorithm 1: Pseudocode for MM

```
1  g_F(start) := g_B(goal) := 0; Open_F := {start};
   Open_B := {goal}; U := ∞
2  while (Open_F ≠ ∅) and (Open_B ≠ ∅) do
3      C := min(prmin_F, prmin_B)
4      if U ≤ max(C, fmin_F, fmin_B, gmin_F + gmin_B + ε)
       then
5          return U
6      if C = prmin_F then
7          // Expand in the forward direction
8          choose n ∈ Open_F for which pr_F(n) = prmin_F
           and g_F(n) is minimum
9          move n from Open_F to Closed_F
10         for each child c of n do
11             if c ∈ Open_F ∪ Closed_F and
               g_F(c) ≤ g_F(n) + cost(n, c) then
12                 continue
13             if c ∈ Open_F ∪ Closed_F then
14                 remove c from Open_F ∪ Closed_F
15             g_F(c) := g_F(n) + cost(n, c)
16             add c to Open_F
17             if c ∈ Open_B then
18                 U := min(U, g_F(c) + g_B(c))
19     else
20         // Expand in the backward direction, analogously
21 return ∞
```

respectively, and $C = min(prmin_F, prmin_B)$. On each iteration MM expands a node with priority C. U is the cost of the cheapest solution found so far. Initially infinite, U is updated whenever a better solution is found. MM stops when

$$U \leq \max(C, fmin_F, fmin_B, gmin_F + gmin_B + \epsilon)$$

where ϵ is the cost of the cheapest edge in the state-space.

Each of the last three terms inside the max is a lower bound on the cost of any solution that might be found by continuing to search. Therefore, if U is smaller than or equal to any of them, its optimality is guaranteed and MM can safely stop. It is safe to stop when $U \leq C$ because $C \leq C^*$ until all optimal solutions have been found (Theorem 10 in (Holte et al. 2015)). Therefore, $U \leq C$ implies $U = C^*$.

MM has the following properties:
(P1) MM's forward (backward) search never expands a state remote or far from $start$ ($goal$), i.e. its forward and backward searches meet in the middle.
(P2) MM never expands a node whose f-value exceeds C^*.
(P3) MM returns C^*.
(P4) If there exists a path from $start$ to $goal$ and MM's heuristics are consistent, MM never expands a state twice.

Holte et al. (2015) gives complete proofs that MM has these properties. Here we provide a sketch of the proof of P1, MM's distinctive property. $2g(n)$ is larger than $f(n)$ when $g(n) > h(n)$. If n is remote or far from $start$ ($goal$) this makes $pr_F(n) > C^*$ ($pr_B(n) > C^*$). If m is near to $start$ ($goal$) on any optimal path, $pr_F(m) \leq C^*$ ($pr_B(m) \leq C^*$). Thus, an optimal solution will be found before MM expands any node that is remote or far from $start$ and far from $goal$.

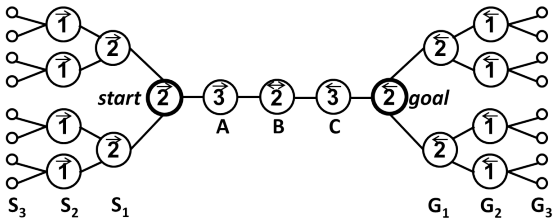

Figure 2: MM$_0$ need not expand all nodes with $g_F(n) < C^*/2$ or $g_B(n) < C^*/2$.

The $2g(n)$ term also guarantees that a node that is remote or far from $start$ ($goal$) but near to $goal$ ($start$) will be expanded by MM's backward (forward) search (if it is expanded at all). The $2g(n)$ term in $pr(n)$ therefore guarantees property P1.

Algorithm 1 gives pseudocode for MM. Lines 2–20 are the usual *best-first search* expansion cycle. Duplicate detection is in line 11. U is updated in line 18 and checked in line 4. Here, only the cost of the optimal path is returned (line 5). It is straightforward to add code to get an actual solution path.

When $prmin_F = prmin_B$ any rule could be used to break the tie (e.g. Pohl (1969)'s cardinality criterion). However, to exploit the $gmin_F + gmin_B + \epsilon$ stopping condition, it is advantageous to continue to expand nodes in the same direction (in line 6 ties are broken in favor of forward search) until there is no longer a tie or $gmin$ in that direction has increased, and to break ties among nodes with the same priority in the chosen search direction in favor of small g.

3.1 MM$_0$

MM$_0$ is the brute-force version of MM, i.e. MM when $h(n) = 0\ \forall n$. Thus for MM$_0$: $pr_F(n) = 2g_F(n)$ and $pr_B(n) = 2g_B(n)$. MM$_0$ is identical to Nicholson's (1966) algorithm except that Nicholson's stops when $U \leq gmin_F + gmin_B$, whereas MM$_0$ stops when $U \leq gmin_F + gmin_B + \epsilon$.

4 MM$_0$ compared to Uni-BS

We now use the region-based framework introduced in Section 2 to analyze under what conditions one type of algorithm will expand more nodes than another. The analysis will be made on a region-by-region basis, since, as we will see, in all cases except Uni-HS vs. Uni-BS no algorithm-type is superior to any other in all regions. We will summarize these analyses with three general rules (GR1, GR2, and GR3). These are general expectations, not iron-clad guarantees. There are many factors in play in a given situation, each factor may favor a different algorithm-type. It is the net sum of these factors that ultimately determines which algorithm-type outperforms another. Our general rules state what we expect will usually be the dominant forces.

We begin by analyzing the brute-force algorithms since this lays the foundation for the subsequent comparisons.

Uni-BS only expands nodes that are near to or far from $start$. We write this as the equation:

(Eq. 1) Uni-BS = NF + NN + F′N + F′F

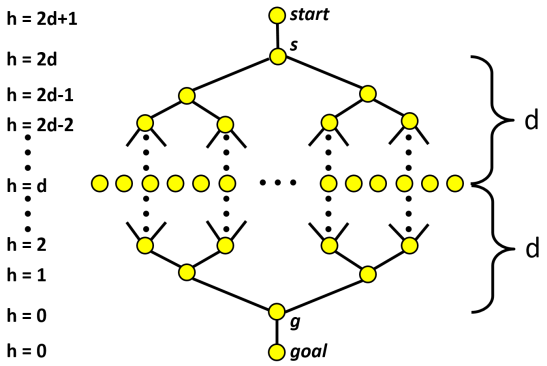

Figure 3: State space in which NN is large.

F′ indicates that Uni-BS might not expand all the nodes that are far from $start$. For example, Uni-BS will usually not expand all nodes that are exactly distance C^* from $start$. By contrast, Uni-BS must expand all nodes near to $start$.

MM$_0$ only expands nodes that are near to $start$ or to $goal$:

(Eq. 2) MM$_0$ = N′F + N′N′ + FN′ + RN′.

N′ indicates that MM$_0$ might not expand all the nodes that are near to $start$ or $goal$. For example, in unit-cost spaces when C^* is even, MM$_0$ will not expand any node in NN because an optimal path will be known by the time all the nodes distance $C^*/2 - 1$ from $start$ and $goal$ have been expanded. The ϵ in the $gmin_F + gmin_B + \epsilon$ termination condition means that MM$_0$ can terminate before some nodes with $g_F(n) < C^*/2$ or $g_B(n) < C^*/2$ have been expanded. This is illustrated in Figure 2; the numbers in the nodes are discussed in Sections 5 and 7, they may be ignored for now. S_i (G_i) is the layer of nodes at depth i in the tree rooted at $start$ ($goal$). After MM$_0$ expands $start$ and $goal$, A and S_1 will be in $Open_F$, and C and G_1 will be in $Open_B$, all with $g = 1$. Since ties are broken in favor of the forward direction, MM$_0$ will next expand A and S_1, generating B and S_2 with $g_F = 2$. It will then switch directions and expand C and G_1 in some order. As soon as C is expanded a solution costing $U = 4$ is found. Since $gmin_F + gmin_B + 1 = 2 + 1 + 1 \geq U$, MM$_0$ can stop. This may happen before some nodes in G_1 are expanded even though they are distance 1 from $goal$ and $C^*/2 = 2$.

Uni-BS expands more nodes than MM$_0$ iff (Eq. 1 > Eq. 2)

(Eq. 3) NF + NN + F′N + F′F > N′F + N′N′ + FN′ + RN′

To identify the *core differences* between the algorithms, i.e. regions explored by one algorithm but not the other, we ignore the difference between N and N′ and between F and F′, which simplifies Eq. 3 to:

(Eq. 4) FF > RN

We have identified two conditions that guarantee FF > RN:

(1) When $C^* = D$, the diameter of the space, there are no remote states, by definition, so RN is empty.

(2) When the number of states far from $start$ is larger than the number of states near to $goal$, i.e. if FF + FN > FN + NN + RN, or equivalently, FF > RN + NN. We say a problem ($start$, $goal$) is *bi-friendly* if it has this property.

A special case of bi-friendly problems occurs when the

number of states distance d from $start$ is the same as the number of states distance d from $goal$, for all $d \leq C^*$. This occurs often in standard heuristic search testbeds, e.g. the Pancake Puzzle, Rubik's Cube, and the Sliding Tile Puzzle when the blank is in the same location type (e.g. a corner) in both $start$ and $goal$. In such cases, a problem is bi-friendly if the number of states near to $start$ is less than the number of states far from $start$, i.e. more than half the states at depths $d \leq C^*$ occur after the solution midpoint. This is similar to the condition in BK1 with $h(s) = 0 \forall s$. In many testbeds this occurs because the number of states distance d from any state continues to grow as d increases until d is well past $D/2$. For example, Rubik's Cube has $D = 20$ and the number of states at distance d only begins to decrease when $d = 19$ (Table 5.1, (Rokicki et al. 2013)).

Non-core differences (NF, NN, FN) can sometimes cause large performance differences. The example in Figure 3 exploits the fact that Uni-BS always expands all nodes in NN but $\texttt{MM}_0$ sometimes does not. All edges cost 1. $start$ and $goal$ each have one neighbor (s and g respectively) that are roots of depth d binary trees that share leaves (the middle layer, which is NN). $C^* = 2d + 2$ and all paths from $start$ to $goal$ are optimal. FF and RN are empty. The values on the figure's left may be ignored for now, they are used in Section 5. $\texttt{MM}_0$ expands all the nodes except those in the middle layer, for a total of $2 \cdot 2^d$ nodes expanded. Uni-BS will expand all the nodes except $goal$, for a total of $3 \cdot 2^d - 1$ nodes, 1.5 times as many as $\texttt{MM}_0$. This ratio can be made arbitrarily large by increasing the branching factor of the trees.

The general rule based on the core differences is:

GR1: FF and RN usually determine whether $\texttt{MM}_0$ will expand fewer nodes than Uni-BS (FF $>$ RN) or more.

5 $\texttt{MM}_0$ compared to A*

A heuristic, h, splits each region into two parts, the states in the region that are pruned by h, and the states that are not pruned. For example, FNU is the unpruned part of FN. The set of states expanded by A* is therefore (modified Eq. 1):

(Eq. 5) A* = NFU + NNU + FNU + FFU

We first compare the first three terms to the corresponding terms in Eq. 2 for $\texttt{MM}_0$ and then compare FFU and RN′.

Region NF: We expect A* to expand many nodes in NF. These nodes have $g_F(n) \leq C^*/2$ so A* would prune them only if $h_F(n) > C^*/2$. One might expect $\texttt{MM}_0$'s N′F to be larger than A*'s NFU because A* prunes NF with a heuristic. This underestimates the power of the $gmin_F + gmin_B + \epsilon$ termination condition, which can cause N′F to be much smaller than NFU. In Figure 2, a number with a right-pointing arrow over it inside node n is $h_F(n)$. Not shown are $h_F(C) = 1$ and $h_F(s) = 1 \ \forall s \in S_3$. Region NF contains $start$, A, S_1 and S_2. The heuristic does no pruning in this region so these are all expanded by A*. $\texttt{MM}_0$ will not expand any node n with $g_F(n) = C^*/2$ (e.g. S_2) so N′F is half the size of NFU. As a second example, on Rubik's Cube instances with $C^* = 20$, $\texttt{MM}_0$ only expands nodes with $g_F(n) \leq 9$ because of this termination condition. Korf (1997)'s heuristic has a maximum value of 11, so A* with this heuristic will not prune any nodes in N′F. In general, we do not expect A* to have a large advantage over $\texttt{MM}_0$ in NF unless its heuristic is very accurate.[2]

Region NN: As discussed above, $\texttt{MM}_0$ might expand no nodes in NN (i.e. N′N′ is empty). Nodes in NN have $g_F(n) = g_B(n) = C^*/2$, so A*'s $f(s)$ cannot exceed C^*. Therefore, even with an extremely accurate heuristic, A* may do little pruning in NN. For example, the heuristic values shown on the left side of Figure 3 are consistent and "almost perfect" (Helmert and Röger 2008) yet they produce no pruning at all. A* behaves exactly the same as Uni-BS and expands 1.5 times as many nodes as $\texttt{MM}_0$.

Region FN: We expect A* to expand far fewer nodes than $\texttt{MM}_0$ in FN. These nodes have $g_F(n) > C^*/2$ and, being relatively close to $goal$, we expect the heuristic values for these nodes to be very accurate.

FFU vs RN′: RN′ certainly can be much smaller than FFU. In Figure 2, RN ($G_1 + G_2$) is about the same size as FF (S_3), which is the same as FFU in this example. However, because $\texttt{MM}_0$ will not expand any nodes with $g_B(n) = C^*/2$ in this example, RN′ is half the size of RN (RN′ contains G_1 but not G_2), so $\texttt{MM}_0$ expands many fewer nodes in RN than A* does in FF. On the other hand, FFU will certainly be the same size as or smaller than RN′ with a sufficiently accurate heuristic. In the extreme case, when RN′ is empty, this requires a heuristic that prunes every node in FF. This is not impossible, since no optimal path passes through FF, but it does require an extremely accurate heuristic. Moreover, FF without any pruning can be much smaller than RN′. Deleting S_3 from Figure 2 makes FF empty, while RN′ can be made arbitrarily large.

The general rule based on the core differences is:

GR2: When FF $>$ RN, A* will expand more nodes in FF than $\texttt{MM}_0$ expands in RN unless A*'s heuristic is very accurate.

6 MM compared to A*

Modifying Eq. 2, the equation for MM is:

(Eq. 6) MM = N′FU + N′N′U + FN′B + RN′B.

B has the same meaning as U, but is based on h_B, the heuristic of MM's backwards search. For example, FNB is the part of FN that is not pruned by h_B. In general, FNB will be different than FNU, the part that is not pruned by h_F, the heuristic used by A*. By definition, N′FU $\leq$ NFU and N′N′U $\leq$ NN, so MM has an advantage over A* in NF and NN.

Region FN: FNU is almost certainly smaller than FN′B because in forward search, nodes in FN have $g_F(n) > C^*/2$ and h_F is estimating a small distance (at most $C^*/2$). By contrast, for the backwards search, nodes in FN have $g_B(n) \leq C^*/2$ and h_B would need to accurately estimate a distance larger than $C^*/2$ to prune them. So, A* has an advantage over MM's backward search in FN.

[2] We recognize the imprecision in terms like "very accurate", "inaccurate" etc. We use these qualitative gradations to highlight that as the heuristic's accuracy increases or decreases, the advantage shifts from one algorithm to another.

FFU vs RNB: Not much pruning will usually occur during MM's backward search in RN because RN's g_B-values are small and the distances being estimated by h_B are large. However, if RN is much smaller than FF but h_F is accurate enough to make FFU smaller than MM_0's RN′ (see the discussion of FFU vs RN′ in Section 5), then we expect that h_B will be accurate enough to do some pruning in RN. Thus, we expect FFU > RNB whenever RN is much smaller than FF.

The general rule based on this section's analysis is the same as GR2 with MM_0 replaced by MM.

6.1 The Correctness of BK1

In our notation BK1 (page 1) is written as:

FNU + FFU < NNU + NFU $\implies$ Uni-HS < MM.

Here FNU + FFU is the number of nodes expanded by Uni-HS beyond the solution midpoint, NNU + NFU is the number expanded at or before the solution midpoint.

Combining Eqs. 5 and 6 gives the exact expression:

(Eq. 7) Uni-HS < MM $\iff$ NFU + NNU + FNU + FFU < N′FU + N′N′U + FN′B + RN′B.

Differences between Eq. 7 and BK1 represent situations in which BK1 will be incorrect. For example, BK1 ignores region RN, but it can be the decisive factor determining whether MM expands more nodes than Uni-HS.

On the other hand, it is easy to prove (Holte et al. 2015) that BK1 will make correct predictions if the following conditions all hold: **(C1)** NN, FNU, and FFU are all negligible in size compared to NFU; **(C2)** FN′B + RN′B ≈ N′FU; **(C3)** NFU/2 < N′FU.

C1–C3 hold in our experiment on the Pancake Puzzle with the GAP heuristic (see the GAP rows for A* and MM near the bottom of Table 1) and we conjecture they held in Barker and Korf's experiments.

7 MM_0 compared to MM: an anomaly

If h_1 and h_2 are admissible heuristics and $h_1(s) > h_2(s)$ for all non-goal nodes, then A* cannot expand more distinct nodes with h_1 than with h_2 (Nilsson 1982). In particular, A* with a non-zero heuristic cannot expand more nodes than Uni-BS.

This is not necessarily true for MM or most Bi-HS algorithms. In Figure 2 the value in a node is its h-value in the direction indicated by the arrow. All nodes in layer S_3 (G_3) have $h_F(s) = 1$ ($h_B(s) = 1$). MM expands all the nodes in S_1 and G_1 because they have $pr(s) = 3$ while $pr_F(A) = pr_B(C) = 4$. MM might then expand any number of nodes in S_2 or G_2 since they too have $pr(s) = 4$.[3] By contrast, we saw (Section 4) that MM_0 could stop before expanding all the nodes in S_1 and G_1 and would never expand a node in S_2 or G_2. Thus we see that MM_0 can expand strictly fewer nodes than MM with a consistent, non-zero heuristic.

This example mimics behavior we saw with the GAP-2 and GAP-3 heuristics in the Pancake puzzle experiments below. We believe it occurs commonly with heuristics that are very accurate near the goal but inaccurate elsewhere. This example reveals a fundamental dilemma for Bi-HS caused by a tension between its two main stopping conditions:

S1: $U \leq \max(fmin_F, fmin_B)$

S2: $U \leq gmin_F + gmin_B + \epsilon$.

To satisfy S1 as quickly as possible, a node with minimum f-value should be expanded, but to satisfy S2 as quickly as possible a node with minimum g-value should be expanded. These two node-selection rules will disagree if none of the nodes with the smallest f-value also have the smallest g-value. Breaking ties among nodes with the same f-value in favor of large g-values also causes the two selection rules to make different choices. When the two selection rules disagree, a choice has to be made as to which stopping condition to favor. MM and all previous Bi-HS methods are hard-coded to favor S1. Bi-BS methods must favor S2, since they have no heuristic. In situations like Figure 2, where S2 can be satisfied more quickly than S1, Bi-BS will outperform Bi-HS. Identifying conditions under which S2 is more quickly satisfied than S1 is an important direction for future research. For now, we offer the following conjecture:

GR3: With an inaccurate heuristic, Bi-HS will expand more nodes than Bi-BS.

8 Experiments

The purpose of the experiments in this section is to verify the correctness our general rules (GR1–GR3). Since some rules refer to the sizes of certain regions, they could only be tested in domains small enough to be fully enumerated. Likewise, since some rules refer to a heuristic's relative accuracy, we used at least two heuristics of different accuracy in each domain. All heuristic used in these experiments were consistent, not just admissible. The two domains used in our study are the 10-Pancake Puzzle and Rubik's Cube. In both domains all problems are bi-friendly. Because GR1–GR3 make predictions about the number of nodes expanded, that is the only quantity we measure in our experiments.

8.1 10-Pancake Puzzle

We ran MM_0, MM, Uni-BS, and A* on 30 random instances for each possible value of C^* ($1 \leq C^* \leq 11$). We used the GAP heuristic (Helmert 2010) and derived less accurate heuristics from it, referred to as GAP-X, by not counting the gaps involving any of the X smallest pancakes. For example, GAP-2 does not count the gaps involving pancakes 0 or 1.

The trends reported below were similar for all values of C^*. In addition, similar trends were obtained using a pattern database (PDB) based on 6-X pancakes. Table 1 shows the number of nodes expanded in each region for each algorithm using each heuristic for $C^* = 10$.[4] Row "Reg. Size" shows the number of states in each region. Column "Total" is the total of all the columns to its right. The total for Region Size does not include region RF (it is not in the table because none of the algorithms expand nodes in RF).

[3] Bi-HS algorithms that strictly alternate search direction or use the cardinality criterion to choose the direction will expand all the nodes in S_2 and G_2.

[4] We present results for $C^* = 10$ because the trends reported below were valid for *all* 30 instances with $C^* = 10$. There were a few instances for $C^* \in \{6, 7, 9\}$ that were exceptions. But, most trends were clearly seen for these values of C^* too.

	Total	NF	NN	FF	FN	RN
Reg. Size	3,555,955	27,390	55	3,501,120	27,003	387
Brute-force searches						
Uni-BS	1,743,548	27,390	55	1,704,027	12,077	0
$\texttt{MM}_0$	**5,551**	4,620	0	0	917	14
GAP-3						
A*	97,644	17,346	55	75,431	4,812	0
MM	7,507	4,095	0	0	3,402	11
MM−2g	106,539	17,446	55	78,738	10,289	11
GAP-2						
A*	27,162	9,964	55	14,191	2,952	0
MM	6,723	3,311	0	0	3,402	11
MM−2g	39,453	10,255	55	19,542	9,590	11
GAP-1						
A*	4,280	2,611	55	852	761	0
MM	**2,448**	1,350	0	0	1,097	1
MM−2g	5,967	2,668	55	1,131	2,113	1
GAP						
A*	**117**	91	12	1	13	0
MM	165	88	0	0	77	0
MM−2g	165	88	0	0	77	0

Table 1: 10 pancake results: average nodes expansions by region for instances with $C^* = 10$.

We see that RN is small and FF is very large. Although we regard GAP-3 as a inaccurate heuristic it does prune almost all the nodes in FF. NF is identical in size to FN+RN because of the symmetry in this space. The asymmetry of $\texttt{MM}_0$'s expansions in NF and FN+RN is because, for $C^* = 10$, $\texttt{MM}_0$ must expand all the nodes with $g(s) = 4$ in one direction but not the other. MM's expansions in these regions are much more balanced. A*'s total is largely determined by NF with the more accurate heuristics, but is determined by FF with the less accurate heuristics. The bold results show that depending on h the algorithm expanding the fewest nodes is A* (GAP), MM (GAP-1), or $\texttt{MM}_0$ (GAP-2, GAP-3).

To examine the effect of the $2g$ term in MM's definition of a node's priority, we ran an altered version of MM, called MM−2g, omitting the $2g$ term in the definition of $pr(n)$, so node n's priority is the usual $f(n)$. We also added code to prevent MM−2g from expanding the same node in both directions. Although not identical to any existing Bi-HS algorithm, we believe MM−2g's results are representative of bidirectional search algorithms that do not meet in the middle. For all of the heuristics, MM−2g expands many nodes in FF and many more nodes than MM in NF, NN, and FN.

GR1, GR2, and GR3 are all confirmed by this experiment.

GR1: For every instance for every value of C^*, FF $>$ RN and $\texttt{MM}_0$ expanded fewer nodes than Uni-BS.

GR2: A* expands more and more nodes in FF as the heuristic becomes less accurate, while MM and $\texttt{MM}_0$ always expand less than half the nodes in RN. This trend holds for individual instances, not just for averages. On all 30 instances A* with the GAP heuristic does not expand more nodes in FF than $\texttt{MM}_0$ expands in RN, and the opposite is true for all instances when A* uses the other heuristics. MM is similar – with all heuristics MM expands fewer nodes in RN than A* does in FF on all instances.

			h_{1997}		h_{888}	
#	d	MM0	MM	IDA*	MM	IDA*
1	16	218M	**166M**	260M	96.0M	**18.7M**
2	17	1.55B	**1.00B**	1.51B	1.01B	**114M**
3	17	1.55B	**1.14B**	8.13B	1.02B	**676M**
4	17	1.55B	**0.96B**	6.56B	**387M**	467M
5	18	**2.88B**	4.71B	29.7B	3.58B	**2.49B**
6	18	**2.88B**	4.84B	15.4B	3.51B	**1.10B**
7	18	**2.88B**	5.89B	41.6B	4.01B	3.16B
8	18	**2.88B**	4.84B	45.9B	3.67B	3.77B
9	18	2.88B	3.01B	58.4B	**2.87B**	5.13B
10	18	**2.88B**	4.25B	70.3B	3.29B	4.82B

Table 2: Rubik's Cube results. M=million, B=billion.

GR3: With the best heuristic, GAP, MM expands many fewer nodes than $\texttt{MM}_0$. As the heuristic becomes less accurate, the difference between MM and $\texttt{MM}_0$ steadily diminishes and eventually (GAP-2) turns into a steadily growing advantage for $\texttt{MM}_0$. This trend holds for individual instances too.

8.2 Rubik's Cube

We use two heuristics in this study: h_{888} is the more accurate, using two 8-edge PDBs and the 8-corner PDB. h_{1997} is the heuristic used to first optimally solve random instances of Rubik's Cube (Korf 1997). It is based on two 6-edge PDBs and the 8-corner PDB.

The Uni-HS algorithm is IDA* with the standard operator pruning rules for Rubik's Cube (Korf 1997). Our implementations of MM and $\texttt{MM}_0$ both use external memory. A full description of these implementations is outside of the scope of this paper, but both algorithms are based on delayed duplicate detection (Korf 2004). Our $\texttt{MM}_0$ implementation expands a full g-layer in one direction, and then removes duplicates and checks for a solution. As a result, it always expands the maximum number of states in a layer before completing. Our MM implementation has priority-based open- and closed-lists stored across two 500GB SSD disks. States with the same priority are found in the same file; before a set of states is expanded, duplicate detection against the closed list is performed. Then, solution detection is performed in parallel to expanding states in the file. Because we only check for solutions when expanding a file, there can be a significant delay between when the full solution is generated and detected. Improving this is an issue for future work. Because operator pruning is, in general, unsafe to use in conjunction with duplicate detection (Holte and Burch 2014), MM and $\texttt{MM}_0$ did no operator pruning.

Table 2 shows the results on each of the ten standard test instances (Korf 1997). $\texttt{MM}_0$ expands fewer nodes than IDA* with h_{1997} on all instances except for instance #2 where there was a very small gap. Due to tie-breaking within the last iteration of IDA*, the differences on instances #1 and #2 are not meaningful for either algorithm. This is consistent with GR2 because h_{1997} is not especially accurate.

With h_{1997} MM always expands fewer nodes than IDA*. In fact, MM with h_{1997} expands fewer nodes than IDA* with the superior h_{888} on instances #9 and #10. MM expands fewer nodes than $\texttt{MM}_0$ on the easier instances ($d = 17$) but more

on the harder ones ($d = 18$). There are two possible explanations for this. The first is the anomaly phenomenon described in Section 7. A heuristic that is sufficiently accurate for MM to expand fewer nodes than MM_0 on easier instances might not be sufficiently accurate on harder instances. The second is related to the delayed duplicate (and solution) detection. If we performed solution detection earlier MM would have certainly improved. But earlier solution detection in MM_0 could also improve its performance. Future work will study termination conditions in external memory search. For instance, an in-memory version of MM expanded only 75M nodes on problem #1, while tie-breaking in the order of file expansion for external-memory MM can significantly worsen its performance. The IDA* code expands more nodes per second than MM, but for instances #3-#10 MM found solutions in less time than IDA*.

h_{888} is accurate enough (as is GAP on the Pancake puzzle) for IDA* to outperform the MM variants for the easier instances #1 ($d = 16$) but the MM variants expand fewer nodes on the harder instances because h_{888} is not sufficiently accurate on them.

9 Conclusions and future work

In this paper we introduced MM, the first Bi-HS algorithm guaranteed to meet in the middle. We also introduced a framework that divides the state-space into disjoint regions and allows a careful analysis of the behavior of the different algorithms in each of the regions. We studied the various types of algorithms and provided some general rules that were confirmed by our experiments.

This paper initiated this direction. Future work will continue as follows: **(1)** A deeper analysis on current and new MM variants may further deepen our knowledge in this issue. **(2)** A thorough experimental comparison should be done on more domains and with more bidirectional search algorithms. **(3)** Heuristics that are specifically designed for MM, i.e., that only return values larger than $C^*/2$ are needed.

10 Acknowledgements

Thanks to Joseph Barker for answering questions and providing extra data related to (Barker and Korf 2015) and to Sandra Zilles and André Grahl Pereira for suggesting improvements in the theoretical analysis of MM. Financial support for this research was in part provided by Canada's Natural Science and Engineering Research Council (NSERC) and by Israel Science Foundation (ISF) grant #417/13. Computational facilities for some of our experiments were provided by Compute Canada. This material is based upon work supported by the National Science Foundation under Grant No. 1551406.

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 *Proc. 16th European Conference on Artificial Intelligence (ECAI)*, 141–145.

Barker, J. K., and Korf, R. E. 2015. Limitations of front-to-end bidirectional heuristic search. In *Proc. 29th AAAI Conference on Artificial Intelligence*, 1086–1092.

Davis, H. W.; Pollack, R. B.; and Sudkamp, T. 1984. Towards a better understanding of bidirectional search. In *Proc. National Conference on Artificial Intelligence (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.

Eckerle, J. 1994. An optimal bidirectional search algorithm. In *Proc. KI-94: Advances in Artificial Intelligence, 18th Annual German Conference on Artificial Intelligence*, 394.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *Proc. 23rd AAAI Conference on Artificial Intelligence*, 944–949.

Helmert, M. 2010. Landmark heuristics for the pancake problem. In *Proc. 3rd Annual Symposium on Combinatorial Search, (SoCS)*.

Holte, R. C., and Burch, N. 2014. Automatic move pruning for single-agent search. *AI Communications* 27(4):363–383.

Holte, R. C.; Felner, A.; Sharon, G.; and Sturtevant, N. R. 2015. Bidirectional search that is guaranteed to meet in the middle: Extended Version. Technical Report TR15-01, Computing Science Department, University of Alberta.

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. *J. Artificial Intelligence Resesearch (JAIR)* 7:283–317.

Kaindl, H., and Khorsand, A. 1994. Memory-bounded bidirectional search. In *Proc. 12th National Conference on Artificial Intelligence (AAAI)*, 1359–1364.

Korf, R. E. 1997. Finding optimal solutions to Rubik's Cube using pattern databases. In *Proc. 14th National Conference on Artificial Intelligence (AAAI)*, 700–705.

Korf, R. E. 2004. Best-first frontier search with delayed duplicate detection. In *Proc. 19th National Conference on Artificial Intelligence (AAAI)*, 650–657.

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

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. 1982. *Principles of Artificial Intelligence*. Springer.

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

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

Rokicki, T.; Kociemba, H.; Davidson, M.; and Dethridge, J. 2013. The diameter of the Rubik's Cube group is twenty. *SIAM J. Discrete Math.* 27(2):1082–1105.

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 COMPUTER SOCIETY of INDIA (CSI-2012)*.