Bidirectional Search That Is Guaranteed to Meet in the Middle: Extended Version

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

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 (see Section 2). 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 empirical evaluation of MM is an important study that we will undertake in the future.

This technical report is an extended version of a AAAI'2016 paper by the same authors entitled "Bidirectional Search That Is Guaranteed to Meet in the Middle". The main additional material is the detailed proofs of all the claims in the AAAI paper.

2 Discussion of Previous Work

MM is the very first bidirectional heuristic search algorithm that is guaranteed to meet in the middle. Papers on traditional front-to-front[1] bidirectional heuristic search typi-

[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

cally claim their searches meet in the middle, but none of them has a theorem to this effect (Arefin and Saha 2010; de Champeaux and Sint 1977; de Champeaux 1983; Davis, Pollack, and Sudkamp 1984; Eckerle 1994; Politowski and Pohl 1984). Perimeter-style front-to-front bidirectional searches with a fixed perimeter size (Dillenburg and Nelson 1994; Kaindl and Kainz 1997; Linares López and Junghanns 2002; Manzini 1995) will only meet in the middle if the middle is known a priori and the perimeter size is chosen accordingly. Wilt and Ruml (2013) describe a perimeter search in which the perimeter size can increase as search progresses, but their aim is to minimize the total number of nodes expanded, not to have the searches meet in the middle.

The only theorem we have found asserting that a bidirectional search meets in the middle is Theorem 5 in an unpublished working paper by Mahanti et al. (2011), but no proof is given and the subsequent publication (Sadhukhan 2012) based on the ideas in this working paper does not include any such theorem.

The only previous paper on front-to-end bidirectional heuristic search with the explicit goal of having the searches meet in the middle describes an algorithm called 2PBS* (Pulido, Mandow, and Pérez de la Cruz 2012). Building on the idea of Kaindl et al. (1999), 2PBS* proceeds in two phases. In the first, a normal front-to-end bidirectional search is executed, but if a state is generated in both directions it is removed from both Open lists and put into a special "frontier" list. This phase ends when one of the Open lists is empty. The second phase then conducts a normal unidirectional heuristic search from the states in the frontier list in order to find an optimal path and prove its optimality. There is no theoretical or experimental evidence given for the two searches in the first phase meeting in the middle and it would fail to do so on the example in Figure 1.

Standard front-to-end bidirectional heuristic searches use a variety of rules to decide which direction to expand next. The simplest (Ikeda et al. 1994; Sadhukhan 2012) is to strictly alternate between the directions so that the number of nodes expanded in the two directions is the same (within one). More commonly, the search direction is chosen based on Pohl's "cardinality" criterion, which expands a node in the direction whose Open list is smaller (Auer and Kaindl 2004; Kaindl and Khorsand 1994; Kwa 1989; Pohl 1969). Barker and Korf (2012) use a variation of this idea, expanding an entire f-level in one direction and then choosing the search direction that expanded fewer nodes the most recent time it was used. Pulido et al. (2012) use the cardinality criterion until the two searches first meet, at which point the search direction is whichever has the larger minimum f-value in its Open list. All of these methods will fail to meet in the middle (node n) on the graph in Figure 1. Auer and Kaindl (2004)'s BiMax-BS$^*_F$ method alternates search direction after completing an entire f-level in one direction. This will meet in the middle when $h(n) = 0$ for all nodes, but it will not do so when nodes with large g-values can have relatively small f-values.

backward search is defined analogously.

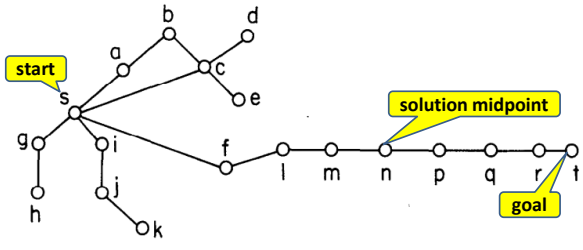

Figure 1: Figure 3.2 in (Pohl 1969). All edges cost 1, $h(n) = 0$ for all nodes.

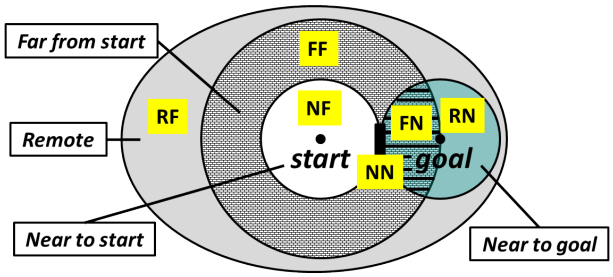

Figure 2: Diagrammatic depiction of the different regions.

3 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 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 2. 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 6–9 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.

3.1 Nodes vs. States

States and nodes are different kinds of entities. A state is an immutable element of a state-space, with a fixed distance to $start$ and $goal$. A node, by contrast, is a mutable entity, created and updated by a search algorithm, representing a path (or set of paths) in the state-space. At a minimum, node n stores the path's cost ($g(n)$) and the last state on the path, which we call the state associated with n.

Rarely, if ever, is there ambiguity about which term should be used in a given context. Nodes are expanded, not states, because the process of expansion requires a g-value and states do not have g-values, only nodes do. Similarly, the regions we define (NN, FF, etc.) are regions of a state-space—sets of states—because they are defined in terms of distances to $start$ and $goal$ and only states have such distances (nodes have g-values).

However, there are a few situations where the correct wording would be awkward. For example, it is technically incorrect to write "how many nodes are expanded in region FF?" The correct way to say this is "how many nodes are expanded whose associated state is in FF". We prefer the simpler expression even though it is not technically correct. Using "node" in a context that requires "state" is harmless because there is a unique state associated with each node. On the other hand, property P4 (below) has the technically incorrect wording "no state is expanded twice". What is meant is that among all the nodes expanded, no two have the same associated state. Likewise, if we say a state s is open (or closed), we mean there is an open (or closed) node whose associated state is s.

4 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 $Open_B$, 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 below). Therefore, $U \leq C$ implies $U = C^*$.

MM has the following properties:

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 ∞
```

(P1) MM's forward and backward searches meet in the middle, i.e. neither search expands a node whose distance from the search's origin ($g_F(n)$ for forward search, $g_B(n)$ for backward search) is larger than $C^*/2$ (Corollary 14 below).

(P2) MM never expands a node whose f-value exceeds C^* (Corollary 14 below).

(P3) MM returns C^* (Lemma 4 below if there is no path from $start$ to $goal$, Theorem 16 if there is).

(P4) If there exists a path from $start$ to $goal$ and MM's heuristics are consistent MM never expands a state twice (Theorem 23 below).

Section 5 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 $goal$. 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. Note that to determine if a better solution path has

been found, MM only checks (line 17) if a newly generated node is in the Open list of the opposite search direction. That is all that is required by our proofs; it is not necessary to also check if a newly generated node is in the Closed list of the opposite search direction. As presented, 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.

4.1 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 $U \leq gmin_F + gmin_B + \epsilon$.

5 Proofs of MM's Properties

In this section we prove that MM has properties P1–P4. We assume that all edge weights are non-negative (zero-cost edges are allowed), that $start \neq goal$, and that the heuristic used by MM in each search direction is admissible. For our analysis in this section we will use the pseudocode in Algorithm 2, which differs from Algorithm 1 in two details.

(1) stopping condition (line 4): For the moment, we will use a weak stopping condition: MM will terminate search as soon as $U \leq C$. This simplifies the proofs of MM's key properties. In Section 5.2 we will replace this stopping condition with the stronger stopping condition used in Algorithm 1 and show that MM maintains all its key properties when the stronger stopping condition is used.

(2) tie-breaking (line 8): To emphasize that the proofs do not depend on how ties among nodes with minimum priority are broken, we have omitted the tie-breaking rule used in Algorithm 1.

The following definition and Lemmas 1–3 are very closely based on Lemma 1 and its proof (for A*) in (Hart, Nilsson, and Raphael 1968).

Definition 1. *Node n is "permanently closed" in the forward search direction if $n \in Closed_F$ and $g_F(n) = d(start, n)$. Likewise, n is permanently closed in the backward search direction if $n \in Closed_B$ and $g_B(n) = d(n, goal)$.*

The name "permanently closed" is based on the following lemma.

Lemma 1. *If node n is permanently closed in a particular search direction at the start of some iteration, it will be permanently closed in that direction at the start of all subsequent iterations.*

Algorithm 2: 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 ≤ C 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
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 ∞
```

Proof. This proof is for the forward search, the proof for the backward search is analogous. There is no code in Algorithm 2 to directly change $g_F(n)$ while $n \in Closed_F$, so n can only stop being permanently closed by being removed from $Closed_F$. This is possible (line 14) but only if a strictly cheaper path to n is found (line 11). This is not possible since $g_F(n) = d(start, n)$ for a node permanently closed in the forward direction. Therefore, once n is permanently closed in the forward direction it will remain so. □

Lemma 2. *Let $P = s_0, s_1, \ldots s_n$ be an optimal path from $start$ (s_0) to any state s_n. If s_n is not permanently closed in the forward direction and either $n = 0$ or $n > 0$ and s_{n-1} is permanently closed in the forward direction, then $s_n \in Open_F$ and $g_F(s_n) = d(start, s_n)$. Analogously, let $P = s_0, s_1, \ldots s_n$ be an optimal path from any state s_0 to $goal = s_n$. If s_0 is not permanently closed in the backward direction and either $n = 0$ or $n > 0$ and s_1 is permanently closed in the backward direction, then $s_0 \in Open_B$ and $g_B(s_0) = d(s_0, goal)$.*

Proof. This proof is for the forward search, the proof for the backward search is analogous. If $n = 0$, $s_0 = start$ has not been closed in the forward direction and the lemma is true because line 1 puts $start \in Open_F$ with $g_F(start) = d(start, start) = 0$. Suppose $n > 0$. When s_{n-1} was expanded to become permanently closed in the forward direction s_n was generated via an optimal path (in lines 11 and 15, $g_F(n) + cost(n, c) = d(start, s_{n-1}) + cost(s_{n-1}, s_n) = d(start, s_n)$). s_n cannot have been permanently closed in the forward direction at that time because if it was, it still would be (Lemma 1). If $s_n \in Closed_F \cup Open_F$ at that time with a suboptimal g-value, then it would have

been removed from $Closed_F \cup Open_F$ (line 14) and added to $Open_F$ (line 16) with $g_F = d(start, s_n)$. If $s_n \notin Closed_F \cup Open_F$ at that time, it would likewise have been added to $Open_F$ with $g_F = d(start, s_n)$ (line 16). Finally, if $s_n \in Open_F$ at that time with $g_F(s_n) = d(start, s_n)$ it would have remained so. Therefore, no matter what s_n's status was at the time s_{n-1} was expanded to become permanently closed in the forward direction, at the end of that iteration $s_n \in Open_F$ and $g_F(s_n) = d(start, s_n)$. In subsequent iterations $g_F(s_n)$ cannot have changed, since that only happens if a strictly cheaper path to s_n is found (lines 11 and 12), which is impossible. It also cannot have been closed, since if that had happened it would now be permanently closed. □

Lemma 3. *Let $P = s_0, s_1, \ldots s_n$ be an optimal path from $start$ (s_0) to any state s_n. If s_n is not permanently closed in the forward direction then there exists an i ($0 \leq i \leq n$) such that $s_i \in Open_F$ and $g_F(s_i) = d(start, s_i)$. Define n_F (for path P) to be the smallest such i. Analogously, let $P = s_0, s_1, \ldots s_n$ be an optimal path from any state s_0 to $goal = s_n$. If s_0 is not permanently closed in the backward direction then there exists an i ($0 \leq i \leq n$) such that $s_i \in Open_B$ and $g_B(s_i) = d(s_i, goal)$. Define n_B (for path P) to be the largest such i.*

Proof. This proof is for the forward search, the proof for the backward search is analogous. If $s_0 = start \notin Closed_F$ then $i = 0$ has the required properties ($start \in Open_F$ and $g_F(start) = d(start, start) = 0$, because of line 1). Suppose $start \in Closed_F$. Let j ($0 \leq j < n$) be the largest index such that s_j is permanently closed. Such a j must exist because $start$ ($j = 0$) is permanently closed. By Lemma 2 $s_{j+1} \in Open_F$ and $g(s_{j+1}) = d(start, s_{j+1})$. Therefore $i = j + 1$ has the required properties. □

Much of the following proof is closely based on Pearl's proof that A* always terminates on finite graphs (Section 3.1.2 (Pearl 1984)).

Lemma 4. *For any finite state-space S with non-negative edge weights* MM *halts for any $start$ and $goal$ states in S. If there is no path from $start$ to $goal$,* MM *returns ∞.*

Proof. If the condition in line 4 is satisfied on some iteration, MM will halt immediately. Suppose the condition in line 4 is never satisfied. Lines 11 and 12 ensure that MM never expands a node via the same path twice and, because there are no negative-cost cycles[2] (non-negative edge weights guarantee this), they also ensure that MM never expands a node via a path containing a cycle. In a finite space there are a finite number of acyclic paths to each state. Therefore each state can only be expanded a finite number of times in each search direction before it becomes permanently closed in that direction, and once it becomes permanently closed in a direction it remains so (Lemma 1). Since each iteration expands a node in one of the search directions, after a finite number of iterations MM will have permanently closed all the nodes reachable in one of the search directions, the Open list for that search direction will be empty, the condition in line 2 for continuing to iterate will not be satisfied, and MM will halt (line 21).

If there is no path from $start$ to $goal$ the condition in line 17 will never be satisfied, so U will always have its initial value of ∞. If C becomes infinite—for example because all $n \in Open_F$ have $h_F(n) = \infty$ indicating that $goal$ cannot be reached from them and all $n \in Open_B$ have $h_B(n) = \infty$ indicating that they cannot be reached from $start$—then the condition in line 4 will be satisfied and MM will return $U = \infty$. If C never becomes infinite, we have shown in the previous paragraph that, after a finite number of iterations, the condition in line 2 for continuing to iterate will not be satisfied, and MM will return ∞ (line 21). □

Definition 2. *If $s \neq start$ and $s \in Open_F \cup Closed_F$ at the start of iteration t with $g_F(s) = g$ then $parent_F(\langle s, t, g \rangle)$ is defined to be the triple $\langle s', t', g' \rangle$ such that on iteration t', s was added to $Open_F$ with $g_F(s) = g$ as a consequence of s' being expanded in the forward direction with $g_F(s') = g - cost(s', s)$. $parent_F(\langle start, t, g \rangle)$ is undefined. Similarly, if $s \neq goal$ and $s \in Open_B \cup Closed_B$ at the start of iteration t with $g_B(s) = g$ then $parent_B(\langle s, t, g \rangle)$ is defined to be the triple $\langle s', t', g' \rangle$ such that on iteration t', s was added to $Open_B$ with $g_B(s) = g$ as a consequence of s' being expanded in the backward direction with $g_B(s') = g - cost(s, s')$. $parent_B(\langle goal, t, g \rangle)$ is undefined.*

Lemma 5. *Suppose $s \neq start$ and $s \in Open_F \cup Closed_F$ at the start of iteration t with $g_F(s) = g$. Then:*
(a) $parent_F(\langle s, t, g \rangle)$ exists,
(b) $parent_F(\langle s, t, g \rangle)$ is unique, and
(c) If $parent_F(\langle s, t, g \rangle) = \langle s', t', g' \rangle$ then $t' < t$.
Likewise, suppose $s \neq goal$ and $s \in Open_B \cup Closed_B$ at the start of iteration t with $g_B(s) = g$. Then:
(a) $parent_B(\langle s, t, g \rangle)$ exists,
(b) $parent_B(\langle s, t, g \rangle)$ is unique, and
(c) If $parent_B(\langle s, t, g \rangle) = \langle s', t', g' \rangle$ then $t' < t$.

Proof. This proof is for the forward direction, the proof for the backward direction is analogous.

(a) If $s \neq start$, the only way it can be added to $Open_F$ is by having been generated by some other node being expanded, and the only way it can be added to $Closed_F$ is to have first been added to $Open_F$.

(b) If a state is added to $Open_F$ multiple times, it must be with a different g-value each time. Therefore s and g together uniquely identify the state (s') that caused s to be added to $Open_F$ with $g_F(s) = g$.

(c) A state cannot be on $Open_F$ or $Closed_F$ with $g_F(s) = g$ until after it has been added to $Open_F$ with $g_F(s) = g$. □

Lemma 6. *Suppose $s \neq start$ and $s \in Open_F \cup Closed_F$ at the start of iteration t with $g_F(s) = d(start, s)$. If $parent_F(\langle s, t, g \rangle) = \langle s', t', g' \rangle$, then s' is permanently closed in the forward direction.*
Likewise, Suppose $s \neq goal$ and $s \in Open_B \cup Closed_B$ at the start of iteration t with $g_B(s) = d(s, goal)$. If $parent_B(\langle s, t, g \rangle) = \langle s', t', g' \rangle$, then s' is permanently closed in the backward direction.

[2]The observation that the proof only requires that there be no negative-cost cycles, as opposed to requiring all edge costs to be non-negative (or positive, as Pearl requires), is due to Gaojian Fan (University of Alberta).

Proof. This proof is for the forward direction, the proof for the backward direction is analogous. $d(start, s) = g_F(s) = g' + cost(s', s) \geq d(start, s') + cost(s', s) \geq d(start, s)$. Therefore all these terms are equal. In particular $g' + cost(s', s) = d(start, s') + cost(s', s)$, i.e. $g' = d(start, s')$. Hence, s' became permanently closed on iteration t' and will remain so for all future iterations (Lemma 1). □

Definition 3. *If $s \neq start$ and $s \in Open_F \cup Closed_F$ at the start of iteration t with $g_F(s) = g$ then the forward generating path for $\langle s, t, g\rangle$, $GenPath_F(\langle s, t, g\rangle)$, is defined recursively:*
$GenPath_F(\langle start, t, g\rangle) = \emptyset$
if $s \neq start$, $GenPath_F(\langle s, t, g\rangle) =$
$GenPath_F(parent_F(\langle s, t, g\rangle)) :: parent_F(\langle s, t, g\rangle)$,
where $X :: Y$ adds element Y to the end of a sequence X. Likewise, if $s \neq goal$ and $s \in Open_B \cup Closed_B$ at the start of iteration t with $g_B(s) = g$ then the backward generating path for $\langle s, t, g\rangle$, $GenPath_B(\langle s, t, g\rangle)$, is defined analogously.

The forward (backward) generating path for $\langle s, t, g\rangle$ is well-defined because the recursion must terminate (t strictly decreases as each recursive call is made, and t cannot be negative) and it cannot terminate at any state other than $start$ ($goal$ for the backward direction) because $parent_F(\langle s, t, g\rangle)$ ($parent_F(\langle s, t, g\rangle)$ for the backward direction) exists for all the $\langle s, t, g\rangle$ generated in this sequence of recursive calls unless $s = start$.

Lemma 7. *Suppose $s \neq start$ and $s \in Open_F \cup Closed_F$ at the start of iteration t with $g_F(s) = d(start, s)$. Then all the states in $GenPath_F(\langle s, t, g\rangle)$ are permanently closed in the forward direction.*
Likewise, suppose $s \neq goal$ and $s \in Open_B \cup Closed_B$ at the start of iteration t with $g_B(s) = d(start, s)$. Then all the states in $GenPath_B(\langle s, t, g\rangle)$ are permanently closed in the backward direction.

Proof. This proof is for the forward direction, the proof for the backward direction is analogous. By Lemma 6, if $parent_F(\langle s, t, g\rangle) = \langle s', t', g'\rangle$ then s' is permanently closed in the forward direction. The same lemma can therefore be applied to $\langle s', t', g'\rangle$ to show that s'' is permanently closed, where $\langle s'', t'', g''\rangle = parent_F(s', t', g')$. This process can be repeated backwards through the entire chain, showing that all states in $GenPath_F(\langle s, t, g\rangle)$ are permanently closed in the forward direction. □

Definition 4. *If $P = s_0, s_1, \ldots s_n$ is an optimal path from $start$ (s_0) to $goal$ (s_n), let i be the largest index such that $s_k \in Closed_F$ $\forall k \in [0, i-1]$, and let j be the smallest index such that $s_k \in Closed_B$ $\forall k \in [j+1, n]$. We say that P "has not been found" if $i < j$ and that P "has been found" otherwise ($i \geq j$).*

Lemma 8. *Let $P = s_0, s_1, \ldots s_n$ be an optimal path from $start(s_0)$ to $goal(s_n)$ that has not been found. Then n_F and n_B, as defined in Lemma 3, both exist for P. Moreover, $n_F = s_i$ and $n_B = s_j$, where s_i and s_j are as defined in Definition 4.*

Proof. Let i and j be as in Definition 4. For the forward search, $s_0, s_1, \ldots s_i$ is an optimal path from $start$ to s_i and $s_i \notin Closed_F$ and therefore is not permanently closed in the forward direction. Therefore, $s_0, s_1, \ldots s_i$ satisfies the conditions of Lemma 3 for the forward direction and $n_F = s_{i'}$ exists for path $s_0, s_1, \ldots s_i$. Because $s_0, s_1, \ldots s_{i-1}$ are all in $Closed_F$, it must be that $i' = i$. Since i' is the smallest index between 0 and i such that $s_{i'} \in Open_F$ and $g_F(s_{i'}) = d(start, s_{i'})$, it is also the smallest index between 0 and n with these properties, so $s_{i'}$ is also n_F for path P. For the backward search, the reasoning is analogous. $s_j, s_{j+1}, \ldots s_n$ is an optimal path from s_j to $goal$ and $s_j \notin Closed_B$ and therefore is not permanently closed in the backward direction. Therefore, $s_j, s_{j+1}, \ldots s_n$ satisfies the conditions of Lemma 3 for the backward direction and $n_B = s_{j'}$ exists for path $s_j, s_{j+1}, \ldots s_n$. Because $s_{j+1}, s_{j+2}, \ldots s_n$ are all in $Closed_B$, it must be that $j' = j$. Since j' is the largest index between j and n such that $s_{j'} \in Open_B$ and $g_B(s_{j'}) = d(s_{j'}, goal)$, it is also the largest index between 0 and n with these properties, so $s_{j'}$ is also n_B for path P. □

Lemma 9. *If $P = s_0, s_1, \ldots s_n$ is an optimal path from $start(s_0)$ to $goal(s_n)$ that has not been found, let $n_F = s_i$ and $n_B = s_j$ be as defined in Lemma 3. Then $g_F(n_F) + g_B(n_B) \leq C^*$.*

Proof. Lemma 8 guarantees that n_F and n_B exist for P. Because edge weights are non-negative and $i < j$, $d(start, s_i) + d(s_j, goal) \leq C^*$, the cost of the whole path P. The lemma follows because $g_F(n_F) = d(start, s_i)$ and $g_B(n_B) = d(s_j, goal)$. □

Theorem 10. *If, at the beginning of an* MM *iteration, there exists an optimal path P from $start$ to $goal$ that has not been found, then $C \leq C^*$.*

Proof. Let n_F and n_B on path P be as defined in Lemma 3. By Lemma 9, $g_F(n_F) + g_B(n_B) \leq C^*$, and therefore at least one of $g_F(n_F)$ and $g_B(n_B)$ must be less than or equal to $C^*/2$. Suppose, without loss of generality, that $g_F(n_F) \leq C^*/2$. Then $pr_F(n_F) \leq C^*$ because $f_F(n_F) \leq C^*$ (because the heuristic h_F is admissible and $g_F(n)$ is optimal) and $g_F(n_F) \leq C^*/2$. Since C is the minimum priority of all the nodes in both Open lists and $n_F \in Open_F$, C cannot be larger than $pr_F(n_F)$ and therefore $C \leq C^*$. □

Lemma 11. *If there exists a path from $start$ to $goal$,* MM *will not terminate until at least one optimal path from $start$ to $goal$ has been found.*

Proof. Lemma 8 guarantees that $Open_F$ and $Open_B$ are both non-empty as long as there is any optimal path from $start$ to $goal$ that has not been found, so the termination condition in Line 2 cannot be satisfied until all optimal paths from $start$ to $goal$ have been found. The only other termination condition is $U \leq C$ (line 4). Assume (for the purpose of contradiction) that this termination condition is satisfied before any optimal path from $start$ to $goal$ has been found. Theorem 10 shows that $C \leq C^*$ until all optimal paths from $start$ to $goal$ have been found, so for $U \leq C$ to hold if no optimal paths from $start$ to $goal$ have been found, U must be equal to C^*. We will now show that $U = C^*$ implies

an optimal path from $start$ to $goal$ has been found, contradicting our assumption, thereby proving the lemma. U is set in line 18. On the iteration in which U was set to C^*, there must have been a child node generated, c that satisfied the conditions of line 17, i.e. $c \in Open_F \cap Open_B$ and $g_F(c)+g_B(c) = C^*$. The latter implies $g_F(c) = d(start, c)$ and $g_B(c) = d(c, goal)$, i.e. c is on an optimal path from from $start$ to $goal$ with optimal g-values in both directions. This means Lemma 7 applies to c in both directions, i.e. that all the nodes on the forward and backward generating paths for c are permanently closed. The concatenation of these two paths, with c in between, is an optimal path from $start$ to $goal$ that was found on the iteration when U was set to C^*. □

Lemma 12. *If there exists a path from start to goal, let $P = s_0, s_1, \ldots s_n$ be the first optimal path from $start(s_0)$ to $goal(s_n)$ that is found during* MM*'s execution, and let $n_F = s_i$ and $n_B = s_j$ be as defined in Lemma 3 at the beginning of the iteration on which P is found. Then during that iteration U will be set to C^* in line 18.*

Proof. Lemma 11 guarantees that P exists, and Lemma 8 guarantees that n_F and n_B exist for P at the beginning of the iteration on which it becomes found. One of them must be expanded on this iteration because P's status will not change from "not found" to "found" if n_F remains on $Open_F$ and n_B remains on $Open_B$. We will complete the proof assuming that n_F is expanded. The proof in the case that n_B is expanded is analogous. We will prove the following before proving the lemma:
(a) When n_F is expanded, n_B will be generated as one of its children;
(b) When the test in Line 11 is applied to n_B ($n_B \in Open_F \cup Closed_F$ and $g_F(n_B) \leq g_F(n_F)+cost(n_F, nB)$) it will fail.

Proof of (a): Suppose n_B is not generated as a child of n_F when it is expanded in the forward direction. Then there must exist one or more nodes between them, i.e. $P = start \ldots n_F\ t_1 \ldots t_k\ n_B \ldots goal (k \geq 1)$. In order for P to be "found" at the end of this iteration, it must be the case that $t_i \in Closed_F\ \forall 1 \leq i \leq k$. Since the path $start \ldots n_F\ t_1$ is an optimal path from $start$ to t_1, $t_1 \in Closed_F$ after being generated by n_F means that it had previously been generated via a different optimal path, which implies an optimal path had previously been found from $start$ to all the t_i and, indeed, to n_B. Combining this previously found optimal path from $start$ to n_B with the optimal path found by the backwards search from n_B to $goal$ creates an optimal path from $start$ to $goal$ that had been found prior to P. This contradicts the premise that P is the first optimal path found from $start$ to $goal$.

Proof of (b): The path $start \ldots n_F\ n_B$ is optimal, i.e. $g_F(n_F)+cost(n_F, n_B) = d(start, n_B)$. The test in line 11 can therefore only succeed if an optimal path from $start$ to n_B had previously been found, which contradicts the premise that P is the first optimal path found from $start$ to $goal$.

Proof of the lemma: Because of (b), the test in line 17 succeeds because n_B is a child of n_F (by (a)) and $n_B \in Open_B$ by definition. Because of (b), $g_F(n_B) + g_B(n_B) = d(start, n_B) + d(n_B, goal) = C^*$, so U will be set to C^* in line 18. □

Lemma 13. *If there exists a path from $start$ to $goal$ and* MM *begins an iteration with $C > C^*$ it will terminate immediately (i.e. without expanding a node on this iteration) and return $U = C^*$.*

Proof. By Theorem 10, $C > C^*$ implies that all optimal solutions have been found, which implies (Lemma 12) $U = C^*$ so the termination criterion in line 4 is satisfied (and it is tested before a node is expanded). □

The following establishes MM's properties P1 and P2.

Corollary 14. MM*'s forward search never expands a node n with $f_F(n) > C^*$ or $g_F(n) > C^*/2$, and* MM*'s backward search never expands a node n with $f_B(n) > C^*$ or $g_B(n) > C^*/2$.*

Proof. If there does not exist a path from $start$ to $goal$, $C^* = \infty$ and nothing can be strictly larger than C^*. If there exists a path from $start$ to $goal$, the proof for the forward search is as follows. The proof for the backward search is analogous. By Lemma 13, MM's forward search never expands a node when $C > C^*$, so if n was expanded in the forward search $pr_F(n) \leq C^*$. Since $pr_F(n) = \max(f_F(n), 2 \cdot g_F(n))$ this means both $f_F(n)$ and $2 \cdot g_F(n)$ are less than or equal to C^*. □

Lemma 15. *If there exists a path P from $start$ to $goal$, $Open_F$ and $Open_B$ are never empty.*

Proof. This is the proof for the forward direction. The proof for the backward direction is analogous. By Lemma 3, for $Open_F$ to be empty all states reachable from $start$ must be permanently closed in the forward direction. This is impossible because $goal$ is reachable from $start$ but, as we will now show, it will never be closed in the forward direction. There are two cases to consider: (1) $C^* > 0$ and (2) $C^* = 0$. Both cases use the fact that for $goal$ to be closed in the forward direction it would first have to be open in the forward direction. Suppose $C^* > 0$. If $goal \in Open_F$, $g_F(goal) \geq dist(start, goal) = C^* > C^*/2$ so, by Corollary 14, it would not be expanded (closed). Alternatively, suppose $C^* = 0$. In this case, $pr_F(s) = 0$ for all states $s \in P$ and therefore $C = 0$ on all iterations, so all the states x expanded in the forward direction will have $g_F(x) = 0$. In particular, for $goal$ to be expanded (closed) in the forward direction, it must first have been added to $Open_F$ with $g_F(goal) = 0$. At the beginning of the iteration that added $goal$ to $Open_F$ with $g_F(goal) = 0$ as a result of expanding node s on path P, path P cannot yet have been "found", for it if had previously been found, U would be 0 and MM would not have executed that iteration because the stopping condition in line 4 would have been satisfied ($U = C = 0$). Since P has not yet been found, $goal$ must still be on $Open_B$with $g_B(goal) = 0$ (line 1) so the test in line 17 will be satisfied and U will be set to 0 in line 18. MM will then terminate (line 4) at the beginning of the next iteration (because $U = C = 0$) without having expanded $goal$ in the forward direction. □

The following establishes MM's property P3.

Theorem 16. *If there exists a path from $start$ to $goal$* MM *returns $U = C^*$.*

Proof. Lemma 15 has shown that MM will never terminate, if there is a path from $start$ to $goal$, by $Open_F$ or $Open_B$ becoming empty, and MM cannot terminate if $C < C^*$, because U cannot be smaller than C^*. Therefore, MM is certain to reach an iteration where $C \geq C^*$. If MM reaches an iteration where $C > C^*$, Lemma 13 guarantees MM will return $U = C^*$. The only reason it might not reach an iteration with $C > C^*$ is that it might terminate on an iteration with $C = C^*$. If termination occurs on such an iteration then we have $U \leq C = C^*$ and therefore $U = C^*$ is returned. □

5.1 MM With Consistent Heuristics

In this section we consider additional properties of MM if its heuristics are consistent.

The following trivial lemma will be used in the proofs of Lemmas 18 and 22.

Lemma 17. *If $a_1 > a_2$ and $b_1 > b_2$ then $\max(a_1, b_1) > \max(a_2, b_2)$.*

Proof. Suppose $\max(a_1, b_1) = a_1$. Then $a_1 \geq b_1 > b_2$. In addition, $a_1 > a_2$ is a premise of the lemma. Together these imply $a_1 > \max(a_2, b_2)$. Combining this with the symmetric argument when $\max(a_1, b_1) = b_1$ we have proven the lemma. □

Lemma 18. *If* MM*'s heuristics are consistent and c is a child of n in the forward search then $pr_F(c) \geq pr_F(n)$. Likewise if* MM*'s heuristics are consistent and c is a child of n in the backward search then $pr_B(c) \geq pr_B(n)$.*

Proof. This proof is for the forward search, the proof for the backward search is analogous. $f_F(c) \geq f_F(n)$ because the heuristic is consistent, and $g_F(c) \geq g_F(n)$ because edge costs are non-negative. Therefore, by Lemma 17, $pr_F(c) = \max(f_F(c), 2 \cdot g_F(c)) \geq pr_F(n) = \max(f_F(n), 2 \cdot g_F(n))$. □

The proof of Lemma 2 requires the ability to re-open closed nodes, and virtually all the results of the previous section depend on that lemma. With consistent heuristics we wish to remove the re-opening of closed nodes from the algorithm, so we now must re-prove the equivalent of Lemma 2 without re-opening closed nodes.

Lemma 19. *If* MM*'s heuristics are consistent then C never decreases from one iteration ($n - 1 \geq 1$) of* MM *to the next (n).*

Proof. Let $n \geq 2$ be any iteration that MM executed beyond line 5 in solving a given problem, let s_n be the node chosen for expansion on iteration n, and X the search direction (forward or backward) used for expanding s_n, i.e. on iteration n s_n was moved from $Open_X$ to $Closed_X$ and its children added to $Open_X$ with no changes being made to the open and closed lists in the other direction. Finally, let $C_n = pr_X(s_n)$ be MM's C value as set in line 3 on iteration n.

If $s_n \neq start$ and $s_n \neq goal$ then s_n was added to $Open_X$ with priority $pr_X(s_n)$ on one or more previous iterations. Let $p < n$ (p for "parent") be the iteration that mostly recently (prior to n) added s_n to $Open_X$ with priority $pr_X(s_n)$. If $p = n - 1$ then $C_n = pr_X(s_n) \geq pr_X(p) = pr_X(s_{n-1}) = C_{n-1}$ follows directly from Lemma 18. If $p < n-1$, then s_n has been on $Open_X$ with its current pr_X-value ever since iteration $p + 1$, so it has been available for expansion, but not selected, on all iterations from $p+1$ up to and including $n - 1$. In particular, it was on $Open_X$ with its current pr_X-value in the most recent iteration $n - 1$, where MM chose to expand a different node s_{n-1} in a possibly different direction Y, instead of expanding s_n in direction X. Since MM chooses a node with the smallest priority on either open list $C_n = pr_X(s_n) \geq C_{n-1} = pr_Y(s_{n-1})$.

Now consider $start$ and $goal$. Before the first iteration begins $Open_F$ is initialized to contain $start$ and $Open_B$ is initialized to contain $goal$. Because $g_F(start) = g_B(goal) = 0$, once these are expanded they will never be added to the open list in that direction again, since 0 is the shortest possible path to them. One of these was expanded on MM's first iteration ($n = 1$). Suppose it was $start$ (analogous reasoning applies if $goal$ was expanded on the first iteration). If $goal$ was never expanded then our proof is complete since it plays no role in determining a C value for any of MM's iterations. If $goal$ was first expanded in the backwards direction on the very next iteration ($n = 2$) then, because MM chooses the node with the smallest priority on either open list we must have $C_2 = pr_B(goal) \geq pr_F(start) = C_1$. If $goal$ was first expanded in the backwards direction on a subsequent iteration, $n > 2$, then similar reasoning to $p < n - 1$ case (above) applies, as follows. $goal$ has been on $Open_B$ with its current pr_B value ever since the first iteration ($n = 1$), so it has been available for expansion, but not selected, on all iterations up to and including $n - 1$. In particular, it was on $Open_B$ with its initial pr_B value in the most recent iteration $n - 1$, where MM chose to expand a different node s_{n-1} in a possibly different direction Y, instead of expanding $goal$ in the backwards direction. Since MM chooses a node with the smallest priority on either open list $C_n = pr_B(goal) \geq C_{n-1} = pr_Y(s_{n-1})$. □

Lemma 20. *Suppose* MM*'s heuristics are consistent. Let $P = s_0, s_1, \ldots s_n$ be an optimal path from $start$ (s_0) to any state s_n. If s_n is not permanently closed in the forward direction and either $n = 0$ or $n > 0$ and s_{n-1} is permanently closed in the forward direction, then $s_n \notin Closed_F$ with $g_F(s_n) > d(start, s_n)$ on any iteration. Analogously, let $P = s_0, s_1, \ldots s_n$ be an optimal path from any state s_0 to $goal = s_n$. If s_0 is not permanently closed in the backward direction and either $n = 0$ or $n > 0$ and s_1 is permanently closed in the backward direction, then $s_0 \notin Closed_B$ and $g_B(s_0) > d(s_0, goal)$ on any iteration.*

Proof. This proof is for the forward search, the proof for the backward search is analogous. If $n = 0$, $s_0 = start$ has not been closed in the forward direction and the lemma is true because $Closed_F$ is initially empty and remains so until $start$ is closed in the forward direction.

Suppose $n > 0$ and let t_1 be the iteration on which s_{n-1} was expanded to become permanently closed in the forward direction. The value of $C = C_{t_1}$ during that iteration was

$pr_F^{t_1}(s_{n-1})$. Because s_n is a child of s_{n-1}, when it was generated on that iteration its priority $pr_F^{t_1}(s_n) \geq pr^{t_1}(s_{n-1})$ (by Lemma 18). If s_n had been on $Closed_F$ with a suboptimal g_F-value prior to iteration t_1 it must have been expanded on an earlier iteration $t_0 < t_1$. The value of $C = C_{t_0} = pr_F^{t_0}(s_n)$ on that iteration, because s_n had only been reached via a suboptimal path, would be strictly greater than $pr_F^{t_1}(s_n)$. Hence $C_{t_0} > C_{t_1}$ even though $t_0 < t_1$, contradicting Lemma 19. So s_n cannot have been on $Closed_F$ with a suboptimal g_F-value prior to iteration t_1. On iteration t_1 it was added to $Open_F$ (if it was not already there) with an optimal g_F-value, so it will never subsequently be added to $Open_F$ with a suboptimal g_F-value, hence it will never subsequently be on $Closed_F$ with a suboptimal g_F-value. □

The following is the equivalent of Lemma 2 when MM has consistent heuristics but does not have the ability to re-open closed nodes.

Corollary 21. *Suppose MM's heuristics are consistent and MM does not re-open closed nodes ("$\cup Closed_F$" is removed from lines 13 and 14). Let $P = s_0, s_1, \ldots s_n$ be an optimal path from $start$ (s_0) to any state s_n. If s_n is not permanently closed in the forward direction and either $n = 0$ or $n > 0$ and s_{n-1} is permanently closed in the forward direction, then $s_n \in Open_F$ and $g_F(s_n) = d(start, s_n)$. Analogously, let $P = s_0, s_1, \ldots s_n$ be an optimal path from any state s_0 to $goal = s_n$. If s_0 is not permanently closed in the backward direction and either $n = 0$ or $n > 0$ and s_1 is permanently closed in the backward direction, then $s_0 \in Open_B$ and $g_B(s_0) = d(s_0, goal)$.*

Proof. The proof of Lemma 2 applies directly since, by Lemma 20, there is no need to test if a newly generated node is on Closed with a suboptimal value. □

Since the proofs of the other lemmas and theorems in the previous section are all based on Lemma 2, because of Corollary 21 they continue to hold when MM has consistent heuristics but does not have the ability to re-open closed nodes.

Lemma 22. *If MM's heuristics are consistent and MM does not re-open closed nodes ("$\cup Closed_F$" is removed from lines 13 and 14), then when MM expands a node its g-value is optimal.*

Proof. This is the proof for nodes expanded in MM's forward search. The proof for its backward search is analogous. Suppose node n has just been added to $Open_F$ (line 16) with a suboptimal cost c, i.e. $n \in Open_F$ with $g_F(n) = c > d(start, n)$. Let P be an optimal path from $start$ to n. Since $n \notin Closed_F$, P satisfies the conditions of Lemma 3 and there exists a node $m = n_F \in Open_F$ on P with $g_F(m) = d(start, m)$. To prove the lemma, all that we need to show is that m will be expanded before n, i.e. that $pr_F(m) < pr_F(n)$. By definition, $pr_F(m) = \max(d(start, m)+h_F(m), 2{\cdot}d(start, m))$ and $pr_F(n) = \max(c + h_F(n), 2 \cdot c)$. By Lemma 17, to show that $pr_F(m) < pr_F(n)$ we only have to show that $d(start, m)+h_F(m) < c+h_F(n)$ and that $2{\cdot}d(start, m) < 2 \cdot c$. The latter follows because edge weights are non-negative, so $d(start, m) \leq d(start, n) < c$. The former follows because the heuristic h_F is consistent, i.e. $h_F(m) \leq d(m, n) + h_F(n)$. This implies $d(start, m) + h_F(m) \leq d(start, m) + d(m, n) + h_F(n) = d(start, n) + h_F(n) < c + h_F(n)$. □

The following establishes MM's property P4.

Theorem 23. *Suppose MM's heuristics are consistent and MM does not re-open closed nodes ("$\cup Closed_F$" is removed from lines 13 and 14). Then if there exists a path from $start$ to $goal$ and , then MM never expands a state twice.*

Proof. MM will not expand a state twice in the same search direction because Lemma 22 guarantees that the first time a state becomes closed it becomes permanently closed. The only remaining possibility for a state to be expanded twice is that it is expanded once in the forward direction and once in the backward direction. If there is such a state, n, by Corollary 14, $g_F(n) \leq C^*/2$ and $g_B(n) \leq C^*/2$. Because C^* is finite and optimal, this implies $g_F(n) = C^*/2$ and $g_B(n) = C^*/2$, i.e. n is a state on an optimal solution path P and $pr_F(n) = pr_B(n) = C^*$. Without loss of generality, suppose n is first expanded in the backward direction. $C = pr_B(n) = C^*$ at the time of this expansion and it cannot decrease as search continues (Lemma 19). By the time n is about to be expanded in the forward direction path P has been found since at that point n has been added to both Open lists. Therefore $U = C^*$, the cost of path P. Thus we have that $U \leq C$ before n is expanded for the second time and therefore MM will terminate before expanding n for the second time. Therefore no state is expanded in both directions. □

5.2 Using A Stronger Stopping Condition

We will now show that MM maintains its four key properties (P1–P4) if it stops as soon as any of the following conditions is true:

1. $U \leq C$ (the stopping condition used above)
2. $U \leq fmin_F$
3. $U \leq fmin_B$
4. $U \leq gmin_F + gmin_B + \epsilon$

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

P3 continues to hold because $fmin_F, fmin_B,$ and $gmin_F + gmin_B + \epsilon$ are all lower bounds on the cost of any solution that might be found by continuing to search. The other properties continue to hold when MM uses the stronger stopping condition because with a stronger stopping condition MM will execute a subset of the iterations it executed with the stopping condition used to prove MM's properties. Since those properties were true of every iteration done with MM's original stopping condition, they are true of every iteration done with the stronger stopping condition.

6 MM$_0$ compared to Uni-BS

We will now use the region-based framework introduced in Section 3 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

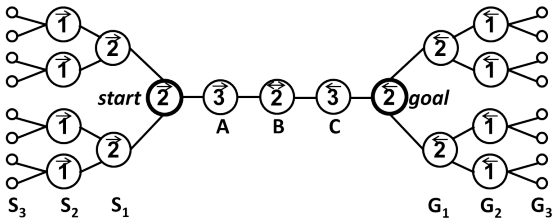

Figure 3: MM_0 need not expand all nodes with $g_F(n) < C^*/2$ or $g_B(n) < C^*/2$.

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, some favoring one algorithm-type, some favoring another. 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

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 3; the numbers in the nodes are discussed in Sections 7 and 9, 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)).

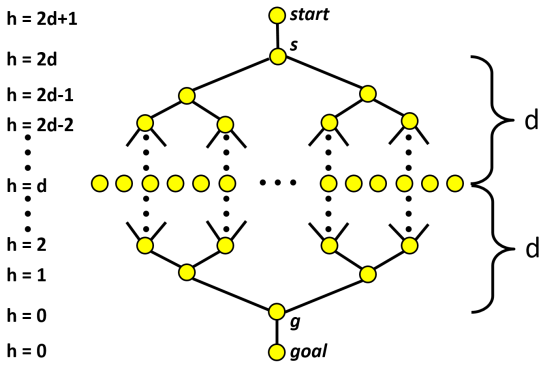

Figure 4: State space in which NN is large.

Non-core differences (NF, NN, FN) can sometimes cause large performance differences. The example in Figure 4 exploits the fact that Uni-BS always expands all nodes in NN but 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 7. 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 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 MM_0 will

expand fewer nodes than Uni-BS (FF $>$ RN) or more.

7 $\texttt{MM}_0$ 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 3, 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.[3]

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 4 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 3, 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 3 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.

8 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.

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 $\texttt{MM}_0$'s RN$'$ (see the discussion of FFU vs RN$'$ in Section 7), 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 $\texttt{MM}_0$ replaced by MM.

8.1 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.

We now show that if all of the following conditions hold, BK1's prediction is correct.

(C1) NN, FNU, and FFU are all negligible in size compared to NFU

(C2) FN$'$B + RN$'$B $\approx$ N$'$FU

(C3) N$'$FU $>$ NFU/2.

[3] 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.

Dropping the negligible terms from the equations above, BK1 becomes

$0 < \text{NFU} \implies \text{Uni-HS} < \texttt{MM}$

i.e. BK1 predicts that Uni-HS $<$ MM always holds under these conditions. The exact analysis simplifies to

$\text{Uni-HS} < \texttt{MM} \iff \text{NFU} < \text{N}'\text{FU} + \text{FN}'\text{B} + \text{RN}'\text{B}.$

C2 says the difference between N′FU and FN′B + RN′B is negligible, so this can be rewritten as

$\text{Uni-HS} < \texttt{MM} \iff \text{NFU} < \text{N}'\text{FU} + \text{N}'\text{FU}$

This inequality is C3, so BK1 is always correct under conditions C1–C3.

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.

9 $\texttt{MM}_0$ 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 every node expanded by A* using h_1 will also be expanded by A* using h_2 (RESULT 6, p. 81 (Nilsson 1982)). In particular, A* with a non-zero heuristic cannot expand more distinct nodes than Uni-BS.[4]

This is not necessarily true for MM or most Bi-HS algorithms. In Figure 3 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 $pr(s) = 4$.[5] By contrast, we saw (Section 6) that $\texttt{MM}_0$ 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 $\texttt{MM}_0$ 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 3, 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.

[4] If the non-zero heuristic is inconsistent, A* may have to re-expand the same nodes many times and, as a result, do more node expansions than Uni-BS even though it expands a subset of the nodes expanded by Uni-BS.

[5] 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.

	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 $C^* = 10$.

10 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.

10.1 10-Pancake Puzzle

We ran $\texttt{MM}_0$, 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 $C^* = 10$.[6] 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).

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.

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.

10.2 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.

			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.

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 9. A heuristic that is sufficiently accurate for MM to expand fewer nodes than $\texttt{MM}_0$ 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 $\texttt{MM}_0$ 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

[6] 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.

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.

11 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.

12 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. 2012. Solving peg solitaire with bidirectional BFIDA*. In *Proc. 26th AAAI Conference on Artificial Intelligence*, 420–426.

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.

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

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

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.

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.

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.

Kaindl, H.; Kainz, G.; Steiner, R.; Auer, A.; and Radda, K. 1999. Switching from bidirectional to unidirectional search. In *Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI)*, 1178–1183.

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.

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

Mahanti, A.; Sadhukan, S. K.; and Ghosh, S. 2011. A tale of two searches: Bidirectional search algorithm that meets in the middle. Technical Report WPS 678, IIM Calcutta.

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

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.

Pearl, J. 1984. *Heuristics – Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

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.

Pulido, F. J.; Mandow, L.; and Pérez de la Cruz, J. 2012. A two-phase bidirectional heuristic search algorithm. In *Proc. 6th Starting AI Researchers Symposium (STAIRS)*, 240–251.

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)*.

Wilt, C. M., and Ruml, W. 2013. Robust bidirectional search via heuristic improvement. In *Proc. 27th AAAI Conference on Artificial Intelligence*, 954–961.