GBFHS: A Generalized Breadth-First Heuristic Search Algorithm

Mike Barley[1], Patricia Riddle[1], Carlos Linares López[2], Sean Dobson[1], Ira Pohl[3]

[1]Computer Science Department, University of Auckland, New Zealand

[2]Computer Science Department, Universidad Carlos III de Madrid, Spain

[3]Computer Science Department, University of California, Santa Cruz

{barley, pat}@cs.auckland.ac.nz, carlos.linares@uc3m.es, seandobs@gmail.com, pohl@soe.ucsc.edu

Abstract

Recently there has been renewed interest in bidirectional heuristic search. New algorithms, e.g., MM, MMe, and NBS, have been introduced which seem much closer to refuting the accepted wisdom that "*any front-to-end bidirectional heuristic search algorithm will likely be dominated by unidirectional heuristic search or bidirectional brute-force search*" . However, MM and MMe can still be dominated by their bidirectional brute-force versions, i.e., they can show a "hump-in-the-middle". We introduce a novel general breadth-first heuristic search algorithm, GBFHS, that unifies both unidirectional and bidirectional search into a single algorithm. It uses knowledge of the edge cost in unit cost domains to stop on first-collision in unidirectional search and in bidirectional search, unlike MM, MMe, and NBS. With no heuristic it expands fewer nodes bidirectionally than Nicholson's blind bidirectional search algorithm. GBFHS expands substantially fewer nodes than MM_0, MM, MMe, and NBS. Additionally, GBFHS does not show a "hump-in-the-middle". GBFHS run bidirectionally is not dominated by bidirectional brute-force search, likewise, GBFHS run unidirectionally is not dominated by A^*.

Introduction

Both bidirectional blind search and heuristic search can exponentially reduce the number of nodes expanded to solve a problem. For almost 50 years now, since Pohl (1969) published his PhD thesis "Bi-Directional and Heuristic Search in Path Problems", researchers have tried to combine bidirectional (blind) search and heuristic search to produce a bidirectional heuristic search algorithm that expands even fewer nodes. The results have largely been disappointing.

Recently there has been renewed interest in bidirectional heuristic search algorithms (Barker and Korf 2012; 2015; Eckerle et al. 2017). New algorithms, MM (Holte et al. 2016; 2017), MMe (Sharon et al. 2016), and NBS (Chen et al. 2017), have been introduced which seem much closer to refuting the accepted wisdom that "*any front-to-end bidirectional heuristic search algorithm will likely be dominated by unidirectional heuristic search or bidirectional brute-force search*" (Barker and Korf 2015). For example, A^* can expand fewer nodes than these algorithms when the heuristic is nearly perfect, but as Figure 1 shows, it quickly starts
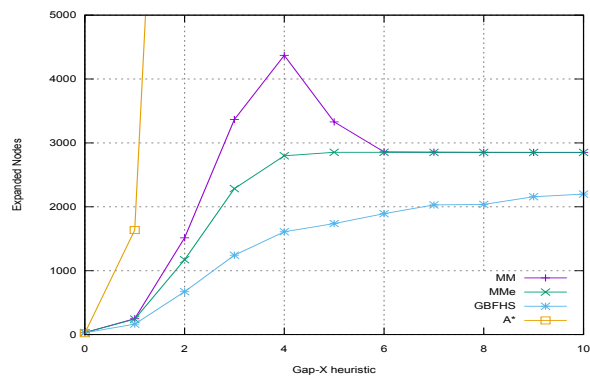
Figure 1: Average number of nodes expanded for A^*, MM, MMe, and GBFHS on 50 random 10-pancake problems as the heuristic accuracy degrades. The GAP-x heuristic ignores the gaps involving the first x discs.

to expand many more nodes than MM, MMe and GBFHS as the heuristic accuracy degrades. Figure 1 shows the average number of expanded nodes for A^*, MM, MMe, and GBFHS on 50 random 10-pancake problems as the heuristic accuracy degrades. For MM and MMe we used our re-implementations taken from their descriptions (Holte et al. 2017), and we also adopted the method they used (Holte et al. 2016)[1] for degrading the GAP heuristic (Helmert 2010). The x-axis in Figure 1 shows the gap heuristic degraded by ignoring the gaps involving the first x discs.

The figure above shows MM is forced to expand nodes that its blind variation does not expand (Holte et al. 2017). In Figure 1, GAP-10 is the 0-heuristic, MM and MMe at GAP-10 are MM_0 while GBFHS at GAP-10 is $GBFHS_0$. We see that MM_0 expands about a third more nodes than $GBFHS_0$. At GAP-4, MM expands about 50% more nodes than MM_0, effectively creating a "hump-in-the-middle", while MMe expands slightly fewer nodes than MM_0. Lastly, at GAP-4, MM expands three times as many nodes as GBFHS and

---

[1]The description of their GAP-x heuristic is ambiguous and our implementation of the GAP-x heuristic may be slightly different than theirs. Our heuristic ignores gaps if either pancake id was less than x and also ignores the table gap if the bottom pancake id was less than x. We found this out too late to rerun the experiments.

MMe expands not quite twice as many nodes as GBFHS. While others have noted this special case (Holte et al. 2017), we are the first to describe the general problem.

We introduce a novel admissible general bidirectional heuristic breadth-first type search algorithm, GBFHS. It has the following improvements over existing algorithms:

1. its frontiers can be made to meet anywhere, it can be made to behave as a forward, backwards, or bidirectional heuristic search algorithm, unlike MM, MMe, and NBS;

2. in unit cost domains, as soon as it finds a solution, it is guaranteed to be optimal whether going unidirectionally or bidirectionally;

3. as a unidirectional algorithm, in unit cost domains, unlike A^*, it does not expand all nodes whose f-values are less than the optimal solution cost;

4. GBFHS is an arbitrary cost algorithm, unlike other breadth-first heuristic algorithms (Zhou and Hansen 2004; 2006);

5. unlike current state-of-the-art bidirectional heuristic search algorithms, it shows no "hump-in-the-middle".

The paper is structured as follows: first, the general problem is presented; next, the related work is introduced. The next section introduces GBFHS and provides evidence for all the claims made above. The paper concludes with various experiments and conclusions.

Ill-Behaved Bidirectional Heuristic Search

One of the main problems with bidirectional heuristic search algorithms is that bidirectional blind search often performs better than heuristic bidirectional search (Edelkamp, Kissmann, and Torralba 2012) producing the hump-in-the-middle phenomenon noted above. This is a special case of a more general problem with these algorithms, namely, that the algorithm can expand more nodes when given a more accurate heuristic than it does with a less accurate one. As this is a counter-intuitive phenomenon, we call these heuristic search algorithms *ill-behaved*. As an example of ill-behaved search algorithms, Holte et al. (2017) "*present an example in which MM, or any Bi-HS algorithm guided by $f(n)$, expands more nodes than MM_0, and witness this occurring in our experiments.*" Ill-behaved heuristic search algorithms can show a hump-in-the-middle on some problems, such as exhibited by MM in Figure 1. The opposite of ill-behaved is *well-behaved*. Well-behaved search algorithms show the expected performance, i. e., to expand fewer nodes as heuristic accuracy improves.

Related Work

Brute-force bidirectional search can be implemented in different ways. The first known algorithm guaranteed to meet in the middle is Nicholson's (1966). However, as observed by Dreyfus (1969), stopping the search when a node is closed in both directions is not sufficient to guarantee optimality, and this algorithm might require additional expansions after the first collision to ensure an optimal solution.

A number of algorithms have tried to use heuristics efficiently in a bidirectional setting. BS^* (Kwa 1989) is an A^*-like bidirectional search algorithm extended with various rules to hasten termination. It performs better than BHPA (Pohl 1971), but sometimes worse than A^*.

Barker and Korf (2015) studied the influence of the heuristic in bidirectional search and claimed that even with a strong heuristic, bidirectional search can expand more nodes than unidirectional search.

MM (Holte et al. 2016; 2017) (from *Meet in the Middle*) is the first bidirectional heuristic search algorithm that is guaranteed to never expand nodes beyond the midpoint. It expands nodes in both directions in ascending order of their *priority*, computed as: $pr(n) = \max\{2g(n), f(n)\}$. When run with the blind heuristic, i.e., $\forall n, h(n) = 0$ (denoted as MM_0), it is equivalent to Nicholson's but with a different termination condition. MM stops only when the cost of the incumbent solution is less or equal to $\max\{pr_{min}, fmin_F, fmin_B, gmin_F + gmin_F + \epsilon\}$, where F/B, and ϵ refer to the forward/backward directions and cheapest operator cost respectively. Even if MM improves the termination condition of Nicholson's, it still can require additional expansions after the first collision and, as shown in Figure 1, it creates a "hump-in-the-middle" when the heuristic accuracy degrades. The frontiers are said to *collide* when nodes containing the same state appear in both frontiers (see Definition 3).

MM was then enhanced by adding $\epsilon(n)$, the cheapest operator applicable to n, to the term $2g(n)$ in the priority formula. The resulting algorithm is denoted as MMe (Sharon et al. 2016). However, it has been proved that neither algorithm dominates the other. Even worse, it is known that both MM and MMe can expand more nodes than MM_0 (Holte et al. 2017), i.e., both MM and MMe are ill-behaved.

NBS (Chen et al. 2017) is based on the fact that any pair of nodes n and m selected from both frontiers that lie on the optimal path satisfy $\max\{f(n), f(m), g(n)+g(m)\} \le C^*$, i.e., it is a lower bound on the optimal cost. It expands at each iteration any pair of nodes with the minimum value of this lower bound. It terminates when this lower bound is greater or equal to the cost of the incumbent solution. Even if it guarantees to never expand more than twice the minimum number of nodes required to optimally solve an instance, it might still be required to expand additional nodes after a first collision occurs (even in unit-cost domains).

Importantly, the theoretical results developed about NBS apply only to bidirectional Deterministic Expansion-based Blackbox (DXBB) algorithms. DXBB algorithms have no a priori information about the instance(s) they are going to solve, nor do they have any knowledge of the heuristic besides that it is admissible (Shaham et al. 2017), and have only black-box access to the *expand*, *heuristic*, and *cost* functions (Chen et al. 2017). This precludes the consideration of either MM or MMe as DXBB search algorithms, as they have an extra piece of information, ϵ, which is defined as the cost of the cheapest operator. As discussed below, GBFHS also has access to ϵ and hence, it is not a DXBB search algorithm. None of these algorithms are therefore directly affected by the theory of Chen et al. 2017.

As we will show in the next Section, GBFHS expands nodes in g-layers. This idea has been tried before, for example, Zhou and Hansen introduced Breadth-First Heuristic Search (Zhou and Hansen 2004; 2006), which also expands nodes in breadth-first order. There are significant differences with GBFHS, other than the fact that their algorithm was specifically designed to reduce the memory requirements of best-first search algorithms, such as A*: on one hand, their algorithm is only applicable to unit-cost domains; on the other hand, to limit the number of nodes generated in each g-layer, they require an upper bound on the cost of the optimal solution which is derived by other means, such as running a beam-search. Instead, GBFHS can be used both in unit and arbitrary-cost domains. In unit-cost domains, both their algorithm and GBFHS halt as soon as they find a solution. Finally, in the next Section we introduce a limit, $fLim$, to better bound the number of nodes in each g-layer, which does not require any pre-computation.

Instead of computing an upper bound on the cost of the optimal solution with beam-search, it is also possible to run Breadth-First Heuristic Search as an IDA* (Korf 1985) with an f-limit that is incremented at each iteration. The resulting algorithm is called BFIDA* (Zhou and Hansen 2004; 2006), and it was improved by Barker and Korf as a bidirectional heuristic search algorithm (Barker and Korf 2012). Basically, the idea consists of running two searches, one from the start state, and another from the goal state using BFIDA*, i. e., in successive iterations, incrementing the f-limit iteratively. Once both frontiers meet, an incumbent solution is found, and the algorithm keeps track of its cost. In their experiments, the optimal solution was always found with a cutoff less than the cost of the incumbent solution. Hence, before starting a new iteration with an f-limit which equals the cost of the incumbent solution, the optimality of this solution can be demonstrated, and the algorithm halts saving the last iteration, the most costly one. While bidirectional BFIDA* was only proved to be effective on Peg Solitaire, GBFHS can be applied to any domain. The main difference is that it does not run in successive iterations, and it uses an OPEN list to select nodes for expansion as any other breadth-first search algorithm.

The GBFHS Algorithm

In this Section, we: (1) describe the GBFHS algorithm; (2) argue that GBFHS is admissible; (3) argue that in unit cost domains, GBFHS can always stop on first collision; (4) show that there are nodes which A* must expand and which the unidirectional version, GBFHS$_F$, does not; and (5) argue that GBFHS is well-behaved.

Our Approach

GBFHS's pseudo-code is shown in Algorithm 1. It uses the function `expandLevel` whose pseudo-code is shown in Algorithm 2 [2]. The inputs to GBFHS are: I, the initial state, G, the goal state, ϵ, the minimum action cost for this domain, and *split*, a function GBFHS uses to determine how far to search in each direction. The pseudo-code shown returns

---

[2]In the pseudocode, $opp(dir)$ is the opposite direction of dir.

Algorithm 1 GBFHS(I,G,ϵ, *split*)→ optimalSolutionCost

1: **if** trivially solved **then**
2:     return(0)
3: best ← unsolvable
4: $open_F \leftarrow \{(I,0)\}$, $open_B \leftarrow \{(G, 0)\}$
5: $closed_F \leftarrow closed_B \leftarrow \varnothing$
6: **for** $fLim$ from $max(h_F(I), h_B(G), \epsilon)$ up by 1 until $open_F = \varnothing \vee open_B = \varnothing$ **do**
7:     **if** best = $fLim$ **then**
8:         return(best)
9:     $gLSum \leftarrow fLim - \epsilon + 1$
10:     $gLim_F, gLim_B \leftarrow$ split($gLSum, gLim_F, gLim_B$)
11:     expandLevel($gLim_F, gLim_B, fLim$)
12:     **if** best = $fLim$ **then**
13:         return(best)
14: return(best)

---

the optimal solution cost, not the actual solution. This simplifies the code but is easy to modify to return the solution. There are various global variables: the open and closed lists (OPEN$_F$, OPEN$_B$, CLOSED$_F$, and CLOSED$_B$), the lowest solution cost found so far, $best$, and the two heuristics, h_F and h_B used in the forward and backward direction respectively. We will first briefly describe some general points and then will look at the algorithm's behavior in various contexts.

GBFHS is a bidirectional arbitrary cost heuristic search algorithm that is a generalization of the standard unidirectional unit cost blind search algorithm. $fLim$ controls GBFHS's exploration of the search space. It is set to the smallest possible solution cost (see line 6 in Algorithm 1) and `expandLevel` (see Algorithm 2) checks whether a solution with that cost (i.e., cost = $fLim$) exists. If none exist then $fLim$ is incremented by one, and this process repeats until either a solution is found with a cost equal to $fLim$ or it determines no solution exists. GBFHS does not expand any node in OPEN$_D$ whose f_D-value is greater than the current $fLim$. The "D" simply indicates the direction of the open list, we use this convention through the rest of this paper.

In addition to the f-value constraint imposed by $fLim$, GBFHS also uses g-value constraints to control the search in each direction, $gLim_F$ and $gLim_B$. GBFHS only expands nodes, n, in OPEN$_D$ where $g_D(n) < gLim_D$. This means that if $gLim_D = 0$ then no node in direction D can be expanded. GBFHS's use of the $gLim$s enables GBFHS to be either a unidirectional or a bidirectional heuristic breadth-first algorithm.

One of the input parameters to GBFHS is ϵ, which is the least cost edge in the search space. ϵ is used to determine the smallest value, $gLSum$, that the two $gLim$s can sum to and still guarantee that all solutions with a cost of $fLim$ have been found by the end of that iteration. Specifically, $gLSum = fLim - \epsilon + 1$.

Another input parameter to GBFHS is the *split* function, which takes three inputs: $gLSum$, $gLim_F$, and $gLim_B$ and returns two values: the updated $gLim$s. The *split* function determines how $gLSum$ is split between the $gLim$s. This determines how much of the search is conducted in the forward

Algorithm 2 expandLevel($gLim_F$, $gLim_B$, $fLim$)

1:   $expandable_F \leftarrow \{n \mid n \in open_F \wedge \text{isExpandable(n,F)}\}$
2:   $expandable_B \leftarrow \{n \mid n \in open_B \wedge \text{isExpandable(n,B)}\}$
3:   **while** $expandable_F \neq \varnothing \wedge expandable_B \neq \varnothing$ **do**
4:      $n \leftarrow \text{pick}(expandable_F \cup expandable_B)$
5:      $dir \leftarrow direction(n)$
6:      $expandable_{dir} \leftarrow expandable_{dir} \smallsetminus \{n\}$
7:      move(n, $open_{dir}$, $closed_{dir}$)
8:      **for all** $c \in \text{expand}(n, dir)$ **do**
9:        **if** $c \in open_{dir} \cup closed_{dir} \wedge g_{dir}(n) + cost(n,c) \geq g_{dir}(c)$ **then**
10:          continue
11:        **if** $c \in open_{dir} \cup closed_{dir}$ **then**
12:          remove($c, open_{dir} \cup closed_{dir}$)
13:        $g_{dir}(c) \leftarrow g_{dir}(n) + cost_{dir}(n,c)$
14:        add($c, open_{dir}$)
15:        **if** isExpandable(c, dir) **then**
16:          add($c, expandable_{dir}$)
17:        **if** $c \in open_{opp(dir)}$ **then**
18:          $best \leftarrow min(best, g_{dir}(c) + g_{opp(dir)}(c))$
19:          **if** $best \leq fLim$ **then**
20:            return
21: return

direction and how much in the backward direction. With an appropriate *split* function, GBFHS can search forward, backward, or any bidirectional split.

We now look at algorithm 1, GBFHS. Line 6 tests whether a collision is no longer possible. If one of the OPEN lists is empty then all nodes in that direction have reached a dead end and the algorithm goes to line 14 where it returns the best solution found so far, if any. $fLim$ is the current lowest bound for the optimal solution cost, i.e., the algorithm has determined that no solution cost can be lower than $fLim$. If the test on line 7 is satisfied then our current best solution, *best*, is an optimal solution.

Definition 1 (*split* Constraints). The *split*($gLSum$, $gLim_F$, $gLim_B$) function must satisfy two constraints: (1) the $gLim$ values it returns must not be lower than the old $gLim$ values supplied to it as parameters, and (2) the sum of the $gLim$ values it returns must equal the value of the $gLSum$ parameter passed to it.

On line 9, $gLSum$ is computed and on line 10 the *split* function is called and divides $gLSum$ between $gLim_F$ and $gLim_B$. The *split* function must satisfy the *split* constraints specified in Defintion 1. On line 11 expandLevel is called to expand all the nodes that are expandable given that $fLim$ and those $gLim$s.

We now look at Algorithm 2, expandLevel. It has three input parameters, $fLim$, $gLim_F$, and $gLim_B$ which determine which nodes in the search space need to be expanded during this iteration. The boolean function isExpandable(n, D) tests whether node n is expandable in direction D. A node n is *expandable* in direction D if $f_D(n) \leq fLim$ and $g_D(n) < gLim_D$ (see Definition 2).

In lines 1 and 2 the set of nodes expandable this iteration in the forward direction, $expandable_F$, and in the backwards

direction, $expandable_B$ are initialized to those nodes, n, that satisfy $f_D(n) \leq fLim$ and $g_D(n) < gLim_D$. Assuming that the search does not terminate during this iteration then all nodes, n, in direction D, that satisfy these conditions, are expanded. Since $fLim$ increases on each iteration, new nodes in both directions can be expanded. However, because $fLim$ only increases by 1, only one $gLim$ is being increased per iteration. Open nodes in the direction where the $gLim$ has not increased, can only be expanded because their expansion had been deferred because their f-value had been too high and now it is not. On line 3, if there are any nodes that are still expandable then this iteration continues. On line 4, we randomly pick a node from the union of the two sets of expandable nodes.

The easiest way to understand our approach is by looking at its simplest context: unidirectional blind search. In this context, *split* always returns a 0 for $gLim_D$ and the search will only go towards D's root node. Let us assume that D is backwards, then the search will be going forwards and the goal G will be the only node in OPEN$_B$. If ϵ is 1 then in line 9 in Algorithm 1, $gLSum = fLim$ and *split* will set $gLim_F = fLim$ and $gLim_B = 0$. Since the heuristics are just returning 0's, a node's f-value is simply its g-value and in unit-cost domains, GBFHS is simply standard breadth-first search. Which guarantees that the first time it generates a goal state it has found an optimal solution path. Otherwise in arbitrary-cost domains, GBFHS is simply standard uniform-cost search and has no such guarantee.

The next easiest context adds a heuristic to unidirectional search. Note that GBFHS does not become any known standard unidirectional heuristic search algorithm. In particular, it does not become textbook A*! In fact, unidirectional heuristic GBFHS can actually expand many fewer nodes than A* and is still guaranteed to be admissible (see Theorem 1). In unit-cost domains, GBFHS (regardless of whether it is going unidirectionally or bidirectionally) can stop on first collision (in unidirectional search this translates into the first goal node being generated) and still be guaranteed to have found an optimal solution (see Theorem 4). Textbook A* has no such guarantees.

The last context is bidirectional blind search, where GBFHS expands one side at a time because there are no deferred nodes in the opposite direction. MM$_0$ can expand from both sides simultaneously. This causes MM$_0$ to have a higher upper bound than blind bidirectional GBFHS, see Figure 3. This results in their expanding more nodes on average. For GBFHS, the order in which the sides are expanded is determined by *split*. Given any order, in unit-cost domains, GBFHS is guaranteed to find optimal solutions on first collision.

Properties of GBFHS

We assume domains with non-negative edge costs. We begin by defining what it means for a node to be expandable in a specific direction.

Definition 2 (Expandable Node). Given $fLim$, and $gLim_D$ where D is a direction. A node n is *expandable* in direction D if $f_D(n) \leq fLim$ and $g_D(n) < gLim_D$.

Definition 3 (Collision). Two nodes *collide* when they are in opposite open lists and have the same state.

Lemma 1 (Expandable Solution Path). *Given a solvable problem P, admissible heuristics h_F and h_B, an optimal solution cost of C^* then an optimal solution path will either be found by GBFHS when $fLim = C^*$ or before.*

Proof Sketch. Given a solution path, S, of cost C^*, since the path cost is simply the sum of the edge costs, for every n in S, $g_F(n) + g_B(n) = C^*$. For $fLim = C^*$, every node n will be expandable in direction D if $f_D(n) \leq fLim$ and $g_D(n) < gLim_D$. A solution path S will be generated as long as at most one node in S is not expanded in either direction. If this is true then S must be generated by the end of the level $fLim = C^*$. Since the heuristics are admissible then the problem cannot be that some node, n, in S has $f(n) > fLim$. When that last level begins, $fLim = gLim_F + gLim_B + \epsilon - 1$. So the only way that a node, n, is not expandable is if $g_F(n) = gLim_F \wedge g_B(n) = gLim_B$.

There are two cases: (1) $\epsilon > 0$, and (2) $\epsilon = 0$. In the first case, since $\epsilon > 0$ every node in S has a different pair of g_F and g_B values. This means that there can be at most one node, n, that has $g_F(n) = gLim_F \wedge g_B(n) = gLim_B$. So, only that node will not be expandable in either direction. Thus, S can be generated.

In the second case, when $\epsilon = 0$, many nodes can have the same pair of g_F and g_B values. However, when $\epsilon = 0$ then $fLim + 1 = gLim_F + gLim_B$. Thus every node, n, in S is expandable because the sum of their g_F and g_B values is C^* but the sum of $gLim$'s is equal to $C^* + 1$, so one of the node's g-values must be less than its corresponding $gLim$. Thus, S can be generated.

Therefore an optimal solution of cost, C^*, will be found on or before $fLim = C^*$. $\square$

Lemma 2 (GBFHS stops when $fLim$ equals optimal solution cost). *If a problem has an optimal solution cost of C^* then GBFHS will stop when $fLim$ equals C^*.*

Proof. From Lemma 1 we know that GBFHS finds an optimal solution before it finishes the $fLim = C^*$ level. On line 12 of Algorithm 1 we see that when a level is done, GBFHS checks whether the cost of the current best solution equals the $fLim$ just finished and if so then stops. Therefore GBFHS will stop when $fLim$ equals C^*. $\square$

Theorem 1 (GBFHS is Admissible). *If P is a solvable problem, h_F, h_B are admissible consistent heuristics and the action costs are non-negative integer valued, with a minimum cost of ϵ, then GBFHS , using heuristics h_F and h_B, will return an optimal solution cost for P.*

Proof Sketch. P is the ordered pair (I, G), where I is the initial state and G is the unique goal state. We show that GBFHS, using heuristics h_F and h_B, will return the optimal solution cost for P. If P is trivially solved, i.e., I equals G, then line 1 in Algorithm 1 returns 0.

If P is not trivially solvable then GBFHS will find the smallest $fLim$ that has a solution, i.e., the optimal solution cost. GBFHS does this by calculating an initial $fLim$ of $max\{(h_F(I), h_B(G), \epsilon)\}$. Because the heuristics are admissible this is guaranteed to be a lower bound on the solution cost. If no solution is found with a cost of this $fLim$,

$fLim$ is increased by one and the search continues in this fashion until $fLim = best$, the cost of the cheapest solution found so far, or one of the open lists is empty. If an open list is empty then there are no more solutions to be found. If $best = unsolvable$ then P is not solvable otherwise $best$ is the optimal solution cost.

Otherwise, GBFHS will search every $fLim$ from $max(h_F(I), h_B(G), \epsilon)$ up to, at most, the optimal solution cost C^*. This process is guaranteed to find a solution with the optimal solution cost as long as GBFHS is guaranteed to always find solutions of cost C^* at or before $fLim = C^*$. Lemma 1 guarantees this, therefore GBFHS is guaranteed to return the optimal solution cost. $\square$

GBFHS$_F$ is when $gLim_B$ is always set to 0.

Theorem 2 (Sufficient Conditions for GBFHS$_F$ Node Expansion). *For any problem with optimal solution cost of C^* and least edge cost of ϵ, GBFHS , with admissible heuristics and the split function always returning $gLim_B$ equal to 0, must expand every node n where $f_F(n) < C*$ and $g_F(n) < C^* - \epsilon$.*

Proof Sketch.

Let C^* be the optimal solution cost and n be an unexpanded node with $f_F(n) < C^*$ and $g_F(n) < C^* - \epsilon$. Since $g_F(n) < C^* - \epsilon$ and the least edge cost is ϵ there could be an ϵ-cost edge connecting n to a goal node and if there were then the cost of that solution would be $g_F(n) + \epsilon$, which would be less than C^*, which contradicts C^* being the optimal solution cost. Therefore all nodes whose f_F-values are less than C^* and whose g_F-values are less than $C^* - \epsilon$, must be expanded to guaranteed that a solution costing C^* is optimal. $\square$

A*'s sufficient conditions for node expansion is $f(n) < C^*$(Dechter and Pearl 1985). By A* we mean the algorithm in textbooks, e. g., AIMA(Russell and Norvig 2016). Specifically, it does not use knowledge of ϵ to control its search. The set of nodes satifsying A*'s sufficient conditions is a superset of the set of nodes satisfying GBFHS$_F$'s sufficient conditions, i. e., using the same tie-breaking order and admissible consistent heuristic, A* can expand a substantially larger number of nodes than GBFHS$_F$. This is because GBFHS$_F$ is using its knowledge of ϵ to reduce the number of nodes that it must expand. This is similar to using the ϵ heuristic (Larsen et al. 2010; Holte 2010) with A* and Fast-Downwards' (Helmert 2006) use of its blind heuristic [3]. It is easy to see that it could be used as a wrapper around any heuristic, however, currently it only seems to be used with the zero heuristic. It returns the larger of ϵ and the original heuristic's value. Also, it would be better to incorporate the use of this knowledge into the algorithm instead of requiring heuristics to be responsible for handling it. For example, A* could incorporate this into its algorithm and either approach, we believe, would provide exactly the same reduction in node expansion as GBFHS's use of ϵ. This allows A* to stop on first generation of a goal in unit cost domains and changes its sufficient conditions to no longer be $f(n) < C^*$ but to become the same as GBFHS$_F$'s.

---

[3]See www.fast-downward.org/Doc/Heuristic\#Blind_heuristic

Now we look at the sufficient conditions for node expansion in the general bidirectional case for GBFHS. The general case is a little more complicated because we need to take into account the split between the $gLim$s and which was the last $gLim$ to be incremented.

Theorem 3 (Sufficient Conditions for GBFHS Node Expansion). *For any problem with optimal solution cost of C^*, least edge cost of ϵ, the last $gLim$ to be expanded being in the D direction (with D' being the opposite direction), and using admissible heuristics, GBFHS must expand every node n where $f_D(n) < C^*$ and $g_D(n) < gLim_D - 1$ and expand every node n where $f_{D'}(n) < C^*$ and $g_{D'}(n) < gLim_{D'} - 1$.*

Proof Sketch.

Let C^* be the optimal solution cost. We look at the conditions on the two directions, D and D', separately. We will first talk about the D direction. Let n be a node where $f_D(n) < C^*$ and $g_D(n) < gLim_D - 1$. Since $g_D(n) < gLim_D - 1$ and since $f_D(n) < C^*$ there could be an ϵ cost edge that connects n to a goal node. If there were such an edge then it would be a solution with a cost less than C^* which would contradict our assumption that C^* was the optimal solution cost. Therefore all nodes whose f_D values were less than C^* and whose g_D values were less than $gLim_D - 1$ must be expanded.

The same reasoning applies for nodes in the D' direction. Consequently, in both cases all such nodes in both directions must be expanded to guarantee that the optimal solution cost is C^*. □

Theorem 4 (In Unit-Cost Domains, GBFHS can stop on First Collision). *If P is a solvable problem in a unit-cost domain, h_F and h_B are both admissible and consistent heuristics, and GBFHS is using h_F and h_B, then the first time that it finds a node with the same state in both open lists, it has found an optimal solution for P.*

Proof Sketch. Assume the optimal solution cost is C^* and all edges are of unit cost, ϵ. Essentially the proof shows that since all edges cost the same, GBFHS cannot find any solution, of cost C, before $fLim = C$. To have a collision between open lists of cost C before $fLim = C$ requires an edge on that solution path with a cost $> \epsilon$. □

This theorem, as seen by the proof sketch, is true whether GBFHS is running bidirectionally or unidirectionally.

Definition 4 (Heuristic domination). A heuristic h_1 is said to *dominate* another heuristic h_2 if and only if $h_1(n) \geq h_2(n)$ for all states n. (Russell and Norvig 2016)

Definition 5 (Stronger and Weaker Heuristic). Heuristic h_1 is *stronger* than heuristic h_2 if h_1 dominates h_2 but h_2 does not dominate h_1. h_2 is *weaker* than h_1.

Definition 6 (Lower-bound). The *lower-bound* for an admissible search algorithm on a problem, P, is the smallest number of nodes that can be expanded to find a solution and guarantee that it is an optimal solution for P.

Definition 7 (Upper-bound). The *upper-bound* for an admissible search algorithm on a problem, P, is the largest number of nodes that can be expanded before finding a solution and guaranteeing that it is an optimal solution for P.

While the idea of a well-behaved heuristic search algorithm seems intuitive, it is hard to nail down. One problem (Dechter and Pearl 1985) is that solving a problem with a weaker heuristic can cause an algorithm to expand fewer nodes than when solving it using stronger heuristic just because of tie-breaking. This is why well-behaved will be defined using lower-bounds which is independent of tie-breaking. Also, in order to compare the behaviour of GBFHS when using two different heuristics, we need to talk about how the heuristics affect the split. The split is defined as GBFHS's final $gLim_F$ and $gLim_B$ values and which direction was incremented last. For example, if using one heuristic, GBFHS finished with both $gLim$s = 12, and with the other it finished with $gLim_F = 24$ and $gLim_B = 0$ then we cannot fairly compare these two runs. Either one could expand fewer nodes depending on the strength of the heuristics. Likewise, if one direction's branching factor is much greater than the other's then if using one heuristic, GBFHS finished with both $gLim$s equal but with one GBFHS ended going forwards while with the other end going backwards, we would expect the one going in the direction of the larger branching factor to have expanded fewer nodes because the penultimate level will be completely expanded but the last might not be. Thus when comparing the behavior of GBFHS with different heuristics it is important that GBFHS finishes solving the problem with the same $gLim$s values and going in the same direction. For each $fLim$ level, GBFHS's *split* function determines these three values.

Definition 8 (GBFHS $gLim$ Split). For GBFHS the $gLim$ *split* for a problem is a trio of values. The first two are the $gLim_F$ and the $gLim_B$ values for the last level and the third value is the direction of the last level's increased $gLim$.

Definition 9 (Well-behaved Bidirectional Algorithm). Let admissible and consistent heuristics h_F^1 and h_B^1 dominate heuristics, h_F^2 and h_B^2 respectively. An algorithm A is said to be *well-behaved* if whenever A stops with the same $gLim$ split using the stronger heuristic pair as when using the weaker pair, there are no problems where A's lower bound, when using the stronger pair, is larger than when using the weaker pair.

Definition 10 (Ill-behaved). An algorithm is *ill-behaved* if there is a problem on which it is not well-behaved.

Theorem 5 (GBFHS guaranteed to be well-behaved). *For any problem P, any front-to-end admissible consistent heuristic pairs h_F^1 and h_B^1, and h_F^2 and h_B^2 such that h_F^1 dominates h_F^2 and h_B^1 dominates h_B^2, and if GBFHS, using either pair of heuristics, stops with the same $gLim$ splits, then GBFHS's lower-bound using the stronger heuristic is never larger than when it uses the weaker heuristic.*

Proof Sketch. Assume S_1 is an optimal solution path that enables GBFHS to expand the least number of nodes using the weaker pair of heuristics, and that this is less than the least number of nodes that GBFHS, using the stronger heuristics, expands. If we use the stronger pair of heuristics on the nodes in S_1 the only change is that some of the nodes may now have a higher f-value, but none of the g-values will change. This means that some nodes expanded using
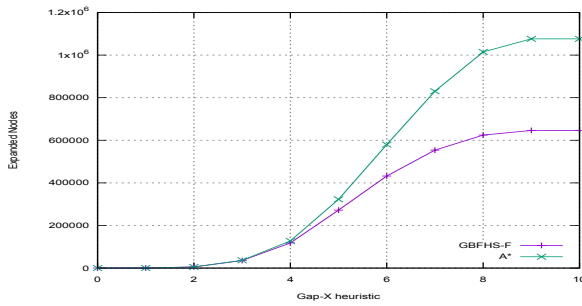
Figure 2: Average Nodes Expanded for A*and GBFHS on 50 random 10-pancake problems as the heuristic accuracy degrades

Figure 3: Actuals, Lower and Upper Bounds for MM, MMe, and GBFHS on 50 random 10-pancake problems as the heuristic degrades.

the weaker heuristics at an earlier level may now be deferred until the last level. At the last level, only those nodes directly on the solution path will be expanded, whereas at earlier levels all those nodes, and possibly more of their descendants, may need to be expanded. So, using a stronger heuristic can only decrease the number of nodes expanded. Consequently, GBFHS is well-behaved. □

Experiments and Analysis[4]

In this section we discuss (1) why GBFHS is better than A*, (2) why GBFHS is better than MM and MMe, and (3) why GBFHS is better than NBS as reported in Chen et al. (2017).

Analysing Why GBFHS Is Better Than A*

It is important to remember that GBFHS run unidirectionally is not A*. Certainly in unit cost domains there can be states that A* has to expand that GBFHS does not. GBFHS does not expand all nodes n where $f(n) < C^*$. Figure 2 shows the average number of nodes expanded by A* and GBFHS run unidirectionally using the same heuristics and same tie-breaking strategy. As the heuristic accuracy degrades, A*starts to expand substantially more nodes than GBFHS run unidirectionally. If you use an "epsilon-enhanced" version of A*, $h(n) = max\{h(n), epsilon\}$, then you will have the same effect as GBFHS and the "epsilon-enhanced" version of A* will also not expand all the nodes, n, where $f(n) < C^*$. Also in unit-cost domains, this enhanced A* would be able to stop on first collision in the same way that GBFHS can. So in summary GBFHS dominates textbook A* and is not dominated by an "epsilon-enhanced" version of A*.

Analysing Why GBFHS Is Better Than MM/MMe

In the Introduction, Figure 1 showed how well GBFHS does against MM and MMe on average over 50 random 10-pancake problems. For the same 50 random problems, Figure 3 shows the lower bounds, actual expanded nodes, and upper bounds for each algorithm. Each algorithm uses their own stylized line. GBFHS is a solid red line, MMe is a
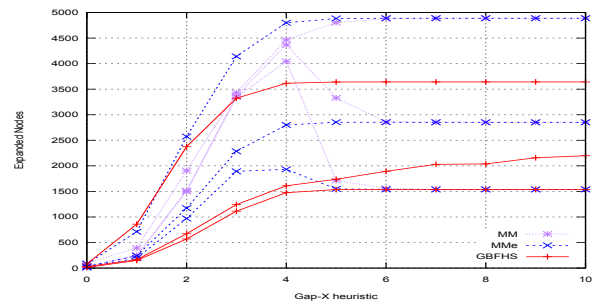
---

[4]The code for running the experiments can be found at https://www.cs.auckland.ac.nz/~barley/SoCS2018GBFHS/.

dashed blue line, and MM is a dotted purple line. The upper lines for each algorithm represents their upper bound, their middle line is their actual node expansions with a particular tie-breaking strategy, and their lower line is their lower bounds. We will refer to these with a LB, UB, TB for lower bound, upper bound, and some choice of tie-break. From now on, we will call the "number of nodes actually expanded" the *actuals*.

At GAP-0, which is very accurate, all 3 algorithms expand the same number of nodes. As the heuristic degrades, the gap between GBFHS's and MM/MMe's widens until at GAP-4, MMe is expanding almost twice as many nodes as GBFHS and MM is expanding almost 3 times. At GAP-6, MM and MMe's number of expanded nodes converge.

As the heuristic degrades, MM exhibits a hump: MM's number of expanded nodes increases until GAP-4 where it peaks at 4300, then it starts decreasing until GAP-6 where it expands about 2900 nodes and stays there, at MM_0's level, until the end. Neither MMe nor GBFHS exhibit such a hump. MMe's expanded nodes increase more slowly than MM's and levels off at GAP-5 where it stays at the MM_0's level until the end. GBFHS's actuals increase the least of all 3 algorithms and never really levels off until it hits the $GBFHS_0$ level, expanding about 2100 nodes. Note that when using GAP-10, i.e., the zero heuristic, MM/MMe expands about a third more nodes than GBFHS. This is one of the things we hope to explain in this section.

Before we discuss the lower and upper bounds, let us discuss how they were calculated for GBFHS, MM, and MMe. Once an algorithm determines it has found a solution and satisfies its stopping criteria, we continue that level until its completion. We calculate the collision sets and stopping sets on the last level. The collision set is the set of sets containing all nodes that could be expanded to get an optimal collision. The stopping set is the set of sets that contains all the nodes that allow the algorithm to terminate. The lower bound is defined as the size of the minimum union of these sets plus the number of nodes expanded on the previous levels. The upper bound is currently estimated as the size of the last level plus the size of all previous levels.

The first possiblity is that GBFHS-LB is lower than MM-LB and/or MMe-LB. Looking at the lower bounds, we note that: (1) MM and MMe's lower bounds are actually higher

domain	heuristic	GBFHS	A*	BS*	MMe	NBS
16 pancake	GAP	279	**125**	339	283	335
16 pancake	GAP-2	**250,941**	1,254,082	947,545	587,283	625,900
16 pancake	GAP-3	**2,140,718**	unsolvable	29,040,138	7,100,998	6,682,497
15 puzzle	MD	12,507,393	15,549,689	**12,001,024**	13,162,312	12,851,889

Table 1: Modified table from NBS paper

than GBFHS-TB until GAP-5; (2) both MM-LB and MMe-LB are ill-behaved, we see them peaking at GAP-4 and then coming back down; and (3) GBFHS-LB is lower than MMe-LB until GAP-5 and GAP-6 for MM-LB, where they converge. After which all lower bounds are identical. So, part of the reason for GBFHS's superior performance is that, at least up to GAP-5, both GBFHS-LB and GBFHS-TB are lower than MM-LB and MMe-LB. Note this rules out the performance difference, up to GAP-5, being due to tie-breaking.

However, even at GAP-5 and beyond, GBFHS-TB still expands many fewer nodes than MM-TB/MMe-TB. Why? To understand that, we need to look at their upper bounds. What we see is that up to GAP-2 for MMe and up to GAP-3 for MM, GBFHS-UB's are worse than MM-UB and MMe-UB. However, from GAP-4 to the end, GBFHS-UB's are substantially lower than those of MM/MMe, 3500 versus 5000, i.e., MM-UB/MMe-UB are almost 50% higher than GBFHS-UB. This is the primary reason for the difference in performance, that, on average, GBFHS has many fewer nodes to explore to find a guaranteed optimal solution. Note this is different than the number of nodes to find a solution or even than finding an optimal solution. Both MM and MMe can find a solution/optimal solution before GBFHS, but often, they still need to satisfy their termination conditions (in unit-cost domains, GBFHS's first collision is guaranteed to be an optimal solution — when it finds a collision the stopping criteria is always satisfied).

So, GBFHS-LB are lower than MM-LB/MMe-LB up to GAP-5 and the same from then on but GBFHS-UB are almost always lower than or equal to MM-UB/MMe-UB. In short, except when the heuristic is very good, e.g., GAP, GBFHS has a much smaller space to explore and consequently, while MM and MMe can sometimes do better than GBFHS, we expect GBFHS's average performance to be better. MM and MMe's use of $max\{f(n), 2g(n)\}$ to calculate a node's priority causes them to expand two GBFHS levels simultaneously, causing them to have a much greater chance of expanding nodes that are not in GBFHS's current search space.

Comparision of GBFHS's Performance to NBS

Chen et al. (2017) report results comparing NBS, A*, GBFHS, MMe, and BS*. We use two of their four domains, namely, the 16 pancake and the 15 puzzle domains to compare GBFHS with NBS. We use their 50 16-pancake problems and Korf's standard 100 problems for the 15 puzzle.

In Table 1, GBFHS always does better than both MMe and NBS. As the heuristic becomes weaker, GBFHS performance becomes better by comparison. For GAP-3, GBFHS does substantially better than both MMe and NBS. NBS ex-

pands more than three times the nodes as GBFHS and MMe is a bit more.

The reader might be suprised that NBS expands more than twice as many nodes as GBFHS, given the theoretical underpinnings of the NBS algorithm. But (Eckerle et al. 2017) states the algorithm must be DXBB. Since GBFHS has knowledge of ϵ it is not a DXBB algorithm so we are not bound by their proof. Neither are MM nor MMe because of their use of ϵ. GBFHS is beaten by A* but this does not refute our earler A* results because this is GBFHS running bidirectionally and meeting in the middle.

Conclusions and Future Research

We introduced, GBFHS, a novel front-to-end bidirectional heuristic search algorithm and showed it is admissible. It is the first bidirectional heuristic search algorithm proved to be well-behaved, i.e., its lower bounds exhibit no hump-in-the middle as the heuristic degrades. Our experiments show GBFHS can expand substantially fewer nodes than NBS, MM, and MMe.

The GBFHS algorithm can run bidirectionally or unidirectionally, with unit-cost or arbitrary-cost, with or without a heuristic. Let us look at each of these cases in turn. When run bidirectionally, GBFHS is not dominated by bidirectional brute-force search. When run unidirectionally, GBFHS dominates textbook A* and is not dominated by "epsilon-enhanced" A*. With unit-cost domains GBFHS can always stop on first collision, regardless of whether running unidirectionally or bidirectionally, whether using a heuristic or not. With arbitrary-cost domains GBFHS cannot necessarily stop on first collision. Without a heuristic, GBFHS is not dominated by brute-force search either unidirectionally nor bidirectionally. GBFHS with an admissible and consistent heuristic is well-behaved and is not dominated by bidirectional brute-force search nor, when run unidirectionally, dominated by A*. These improvements allow unidirectional and bidirectional search to be unified into one general algorithm. By varying the split between the $gLim_F$ and the $gLim_B$, our algorithm behaves as either a unidirectional or bidirectional search algorithm. Assuming a method can be found to automatically determine the split, Barker and Korf's earlier observation will no longer hold.

In Holte's (2010) common misconceptions paper, he states that the following are misconceptions: "In search spaces whose operators all have the same cost A* with the heuristic function $h(s) = 0$ for all states, s, is the same as breadth-first search" and "Bidirectional A* stops when the forward and backward search frontiers meet". While these are misconceptions for A*, the first one is certainly true for GBFHS, and the second one, while not true in general, is

certainly true for GBFHS in unit cost domains.

In the future we will look into the following: (1) Automatically determine the value for ϵ from the problem and domain description; (2) Finding a method for automatically determining a good split; and (3) Investigate arbitrary-cost domains in more depth.

References

Barker, J. K., and Korf, R. E. 2012. Solving peg solitaire with bidirectional BFIDA*. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 420–426.

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

Chen, J.; Holte, R. C.; Zilles, S.; and Sturtevant, N. R. 2017. Front-to-end bidirectional heuristic search with near-optimal node expansions. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 489–495.

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

Dreyfus, S. 1969. An appraisal of some shortest-path algorithms. *Operations Research* 395–412.

Eckerle, J.; Chen, J.; Sturtevant, N.; Zilles, S.; and Holte, R. 2017. Sufficient conditions for node expansion in bidirectional heuristic search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 79–87.

Edelkamp, S.; Kissmann, P.; and Torralba, Á. 2012. Symbolic A* search with pattern databases and the merge-and-shrink abstraction. In *European Conference on Artificial Intelligence (ECAI)*, 306–311.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

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

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

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

Holte, R. 2010. Common Misconceptions Concerning Heuristic Search. In *Third Annual Symposium on Combinatorial Search (SoCS)*, 46–51.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27:97–109.

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

Larsen, B.; Burns, E.; Ruml, W.; and Holte, R. 2010. Searching without a heuristic: Efficient use of abstraction. In *Twenty-fourth AAAI Conference on Artificial Intelligence*, 114–120.

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

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

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

Russell, S. J., and Norvig, P. 2016. *Artificial intelligence: a modern approach*. England; Pearson Education Limited,, 3rd edition.

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

Sharon, G.; Holte, R. C.; Felner, A.; and Sturtevant, N. R. 2016. An improved priority function for bidirectional heuristic search. In *Ninth Annual Symposium on Combinatorial Search (SoCS)*, 139–140.

Zhou, R., and Hansen, E. A. 2004. Breadth-first heuristic search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 92–100.

Zhou, R., and Hansen, E. A. 2006. Breadth-first heuristic search. *Artificial Intelligence* 170(4):385–408.