A Unifying View on Individual Bounds and Heuristic Inaccuracies in Bidirectional Search

Vidal Alcázar
vidal.alcazar@riken.jp
Riken AIP, Tokyo, Japan

Pat Riddle, Mike Barley
{pat, barley}@cs.auckland.ac.nz
University of Auckland, Auckland, New Zealand

Abstract

In the past few years, new very successful bidirectional heuristic search algorithms have been proposed. Their key novelty is a lower bound on the cost of a solution that includes information from the g values in both directions. Kaindl and Kainz (1997) proposed measuring how inaccurate a heuristic is while expanding nodes in the opposite direction, and using this information to raise the f value of the evaluated nodes. However, this comes with a set of disadvantages and remains yet to be exploited to its full potential. Additionally, Sadhukhan (2013) presented BAE*, a bidirectional best-first search algorithm based on the accumulated heuristic inaccuracy along a path. However, no complete comparison in regards to other bidirectional algorithms has yet been done, neither theoretical nor empirical. In this paper we define individual bounds within the lower-bound framework and show how both Kaindl and Kainz's and Sadhukhan's methods can be generalized thus creating new bounds. This overcomes previous shortcomings and allows newer algorithms to benefit from these techniques as well. Experimental results show a substantial improvement, up to an order of magnitude in the number of necessarily-expanded nodes compared to state-of-the-art near-optimal algorithms in common benchmarks.

Introduction

Front-to-end bidirectional heuristic search (Bi-HS) was proposed by Pohl (1969) shortly after A* was presented (Hart, Nilsson, and Raphael 1968). However, Bi-HS was never much better empirically than either A* or blind bidirectional search, and pathologically expanded nodes that blind bidirectional search would not expand despite being more informed (Barker and Korf 2015). Recently, though, the idea of delaying the expansion of nodes with high g addressed this problem partially (Holte et al. 2017; Shaham et al. 2017; 2018) or totally (Barley et al. 2018; Shperberg et al. 2019b). Also, a theoretical analysis on how the g values of nodes from both sides interact (Eckerle et al. 2017) led to the development of algorithms with strong theoretical properties, like NBS's near-optimality (Chen et al. 2017) — meaning that NBS will never expand more than twice the number of necessarily-expanded nodes of any other Bi-HS algorithm.

Reducing the depth of the search is not the only advantage of Bi-HS. Kaindl and Kainz (1997) showed how consistent heuristics can be strengthened by checking how inaccurate they are for nodes in the opposite direction. Naively strengthening them renders them inconsistent, and so far no general method has been widely used in this context.

Another algorithm that uses heuristic inaccuracies is BAE* (Sadhukhan 2013), which adds the estimated error to the target and the heuristic inaccuracy to the source. This value, the total accumulated error, is used both as a criterion to keep consistency (as it depends only on one direction, unlike Kaindl and Kainz's) and to compute a global lower bound. Nevertheless, the relationship to Kaindl and Kainz's methods is unclear, and the empirical evaluation is limited.

An important observation is that both approaches provide global lower bounds. In this paper we delve deeper into the lower-bound framework that modern Bi-HS algorithms use and propose the use of individual lower bounds. These bounds allow integrating recent approaches and the use of heuristic inaccuracies in a simple and general way. Furthermore, bound propagation techniques like those of Shperberg *et.al.* (2019b) can be also seen as a usage of individual lower bounds, with a straightforward way to exploit global bounds for individual nodes too. Empirical results show that, by using heuristic inaccuracies, a substantial increase in performance can be obtained. In fact, we observe that using heuristic inaccuracies allows expanding fewer than half of the necessarily-expanded nodes of near-optimal algorithms, which again emphasizes the importance of the theoretical implications of our approach.

The contributions of this paper are organized by sections: first, we extend the lower-bound framework, introducing the concepts of individual bounds, delayed nodes and relative minimum node values, and proposing a fixpoint computation for the latter; then, Kaindl and Kainz's work is integrated into the framework by creating new bounds, which solves its problematic aspects and finally allows using it to its full potential. Afterwards, Sadhukhan's work is also generalized as an individual bound and linked to the bounds created in the previous section; finally, we present experiments that show a substantial improvement over the state of the art, and outline future lines of research.

Background

A search problem is a tuple $P = (G = (S, E), start, goal, h_f, h_b)$. G is an implicit directed graph; S is the set of vertices (which correspond to states in explicit-state search); E is the set of edges, each of which has a non-negative arbitrary cost; $start \in S$ is the initial state; $goal \in S$ is the goal; h_f, h_b are the forward and backward heuristics respectively. ϵ is the value of the edge with minimum cost, ι is the greatest common denominator among the cost of all non-zero-cost edges, *e.g.* if edges have costs 1.0 and 1.5, $\epsilon = 1.0$ and $\iota = 0.5$.

The cost of the shortest path between two states in S is $c : S \times S \rightarrow \mathbb{R}_{\geq 0}$. Search algorithms keep track of previously seen states using nodes; a node makes reference to a single state, but a single state can be referenced by different nodes. Nodes have node values g, h and f that can be labeled with the direction of the search, like h_f and h_b. When the direction is the same in an equation but can be either one, the label is x, like f_x. Other terms subject to the concept of direction can also be labeled this way, *e.g.* $Open_x$ is the open list of direction x. The opposite direction of x is $\bar{x}$. Given a node n referencing a state $s \in S$, $g_f(n)$ is the cost of the forward path from *start* to n, $g_b(n)$ is the cost of the backward path from *goal* to n, $h_f(n) = h_f(s, goal)$, $h_b(n) = h_b(s, start)$ and $f_x(n) = g_x(n) + h_x(n)$. For $g_f(n)$ and $g_b(n)$ to be optimal, $g_f(n) = c(start, s)$ and $g_b(n) = c(s, goal)$ respectively. h_f and h_b are admissible *iff*, for any state $s \in S$, $h_f(s) \leq c(s, goal)$ and $h_b(s) \leq c(start, s)$ respectively. h_x is consistent *iff* it is admissible and $h_x(n) \leq c(n, n') + h_x(n')$ for any pair of nodes n, n'. The priority function of a best-first search algorithm is a ranking function that maps a node n to a value that determines the order of expansion.

$C^* = c(start, goal)$ is the cost of an optimal solution. U is the cost of the best solution found so far, and thus monotonically decreasing. For an algorithm to be optimal, $C^* = U$ at the end of the execution if there is at least one solution path. C is a lower bound on the cost of any new solution at any given moment, and thus monotonically increasing. When $C \geq U$, necessarily $C^* = U$ and thus an optimal solution has been found. For algorithms to expand nodes with optimal g_x, consistent heuristics are needed. All heuristics in this paper are assumed to be consistent.

Individual Bounds and Relative Minimum Node Values

All consistent optimal algorithms work with C either implicitly or explicitly, *e.g.* in A* (Hart, Nilsson, and Raphael 1968) the implicit C is the minimum f value in the open list. However, keeping track of C explicitly allows implementing more complex techniques (Chen et al. 2017; Barley et al. 2018; Shperberg et al. 2019a; 2019b; Alcázar, Barley, and Riddle 2019). Both Shperberg *et al.* (2019b) and Alcázar *et al.* (2019) propose a high-level algorithm to keep track of C, but the former expands whole layers and the latter defines how states are alternatively expanded. Algorithm 1 defines a more general high-level algorithm that keeps the option of using a last-layer tie-breaking routine like Alcázar *et al.* (2019) but without the specificity of either algorithm.

Algorithm 1 Main Lower Bound Loop

```
1:  U ⇐ ∞; C ⇐ 0
2:  Open_f ⇐ {start}; Open_b ⇐ {goal}
3:  while Open_f ≠ ∅ ∧ Open_b ≠ ∅ do
4:    if UpdateC() then
5:      RunTieBreaker()
6:    end if
7:    if C ≥ U then
8:      return U
9:    end if
10:   Expand()
11: end while
12: return U
```

The purpose of the main loop is to raise C as fast as possible so it converges to U. The purpose of *RunTieBreaker()* is to check whether a solution of cost C exists so U converges to C. *UpdateC()* returns true if C has been increased *i.e.* a higher lower bound has been found.

Eckerle *et al.* (2017) defined a lower bound for any solution path through nodes $n \in Open_f, n' \in Open_b$ such that the cost of the path is at least $max(g_f(n) + g_b(n') + \epsilon, f_f(n), f_b(n'))$. The minimum lower bound among all possible pairs of states from opposite directions is a lower bound on the solution, and thus can be used to increase C. Let us temporarily define $fMin_x$ and $gMin_x$ as the minimum f and g values respectively in $Open_x$; then a global lower bound is $max(gMin_f + gMin_b + \epsilon, fMin_f, fMin_b)$. Because this lower bound takes the maximum of three different equations, and because new bounds will be introduced in subsequent sections, let us individually define the bounds.

Definition 1. *(g bound). The g bound is defined as* $gMin_f + gMin_b + \epsilon$.

Definition 2. *(f bounds). The forward f bound is defined as* $fMin_f$. *The backward f bound is defined as* $fMin_b$.

When $fMin_x$ and $gMin_x$ are indeed the global minimum node values in the open list, C can directly take the value of the highest bound. However, both Shperberg *et al.* (2019b) in Section 4.2 and Alcázar *et al.* (2019) in Theorem 1 pointed out how only nodes such that their f value is equal or lower than C need to be taken into account when computing $gMin_x$, the way NBS (Chen et al. 2017) implicitly does when pairing nodes. In fact, this can be extended to any minimum node value. In order to prove this, let us introduce first the concepts of delayed and expandable nodes.

Definition 3. *(Expandable and delayed node). A node* n *is delayed at layer* C *if it can be proven that no solution path of cost* C *can go through* n. *A non-delayed node is expandable.*

Shperberg *et al.* (2019b) defined a new heuristic $h_{lb}(n)$ based on the lower bound on individual nodes. $h_{lb}(n)$ depends on C and its purpose is in fact to prevent the expansion of the node at a given C. Hence, the obtained numeric value has no bearing other than to be compared with C: if it is higher than C, then delay its expansion until C is increased, turning the node from expandable to delayed.

We now redefine $fMin_x$ and $gMin_x$ as the minimum node values *among expandable nodes*, meaning they are relative to C. Because fewer nodes are considered, higher values for the minimum node values for a given C can be obtained.

Theorem 1. *(Minimum node values only relevant among expandable nodes). When computing minimum node values at a given layer C, only expandable nodes have to be taken into account.*

Proof. (Proof sketch). Let a solution of cost C pass through nodes $n \in Open_x$ and $n' \in Open_{\bar{x}}$. Their lower bound relative to C must not be higher than C, and thus both must be expandable. □

Delayed nodes can also be linked to the concept of individual bounds in a very simple way: for a node generated in direction x, substitute the minimum node values in direction x by the corresponding values of the node. We call this the *delaying rule* of a bound. For instance, a forward node n is delayed by the *g bound* if $g_f(n) + gMin_b + \epsilon > C$; similarly, a backward node n' is delayed by the backward *f bound* if $f_b(n') > C$. However, if the minimum values are relative, C cannot take the value of the global lower bound equation, as admissibility may be compromised. Also, because delaying nodes may cause the minimum node values to increase, and because increasing the minimum node values may cause extra nodes to be delayed, computing the minimum node values and the set of delayed nodes requires a fixpoint computation. Figure 1 illustrates all this.

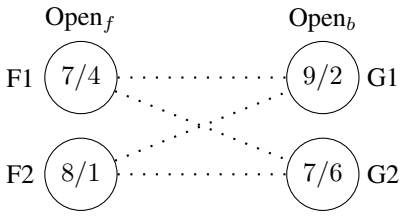

Figure 1: Example of fixpoint computation of minimum node values. First number is f value, second is g value.

Assume $C = 7, \epsilon = \iota = 1$; initially $gMin_f = 1, fMin_f = 7, gMin_b = 2$ and $fMin_b = 7$. The following steps are done:

- F2 and G1 are delayed by the *f bounds*, because $f_f(F2) > C$ and $f_b(G1) > C$. Thus, $gMin_f = 4$ and $gMin_b = 6$.
- The value of the *g bound* is now $gMin_f + gMin_b + \epsilon = 11 > C$, so C must be increased. It would be a mistake to assign 11 to C, as there may be solutions of lower cost *e.g.* a solution of cost 8 through F2 and G2. As all solution costs are a multiple of ι, increasing C by ι means that no C value will be skipped, and thus C is increased to 8 instead. All nodes become expandable again.
- G1 is delayed by the backward f *bound*, because $f_b(G1) > C$. Thus, $gMin_b = 6$.
- F1 is delayed by the *g bound*, because $g_f(F1) + gMin_b + \epsilon > C$. Thus, $fMin_f = 8$.
- No more nodes can be delayed, and the bounds are not greater than C, so the fixpoint computation stops.

In the end, $C = 8, gMin_f = 1, fMin_f = 8, gMin_b = 6$ and $fMin_b = 7$. Note that $fMin_f = 8$ and not 7, which will be relevant once new bounds are introduced. Another criterion to increase C is running out of expandable nodes in either direction instead of checking the value of the bounds. This criterion is equivalent, but checking the bounds gives additional information that may help to decide which criteria should be used as part of the priority function. Following this, we can describe in pseudocode a general way of implementing *UpdateC()* in Algorithm 2 assuming that $fMin_x$ and $gMin_x$ are relative and not global minimum values.

Algorithm 2 UpdateC

```
1: updated ⇐ False
2: while C < max(gMin_f + gMin_b + ε, fMin_f, fMin_b) do
3:     C ⇐ C + ι
4:     updated ⇐ True
5:     UpdateMinValues()
6: end while
7: return updated
```

This fixpoint computation can be expensive. A way of speeding it up is using *g-f* buckets (Burns et al. 2012) and caching results until the addition or removal of a bucket may change the values. Also, if this computation becomes a bottleneck in terms of time, it can be terminated earlier at the expense of having lower minimum node values.

Kaindl and Kainz's Heuristic Inaccuracies

Kaindl and Kainz (1997) observed that, when using consistent heuristics, the lower bound of a node can be increased by exploiting the inaccuracy of the heuristic. This inaccuracy is measured comparing the g value of a node with what the opposite heuristic yields: if a forward node n has $g_f(n) = 3$ and $h_b(n) = 2$, the heuristic inaccuracy of h_b for that node is 1. This value was called $diff_x(n)$, but we will call it just d for succinctness and consistency with other values.

Definition 4. *(d value of a node). The d value of a node n is defined as $d_x(n) = g_x(n) - h_{\bar{x}}(n)$.*

$dMin_x$ is the minimum d value of a set of nodes in direction x. Originally the set of nodes was $Open_x$ and thus $dMin_x$ was non-relative. Two different methods were proposed, the *Add* and the *Max* method, which we will call *KKAdd* and *KKMax*, as other works in the literature have done. *KKAdd* increases the f value of a node by adding the value of *dMin* in the opposite direction, that is, $KKAdd_x(n) = f_x(n) + dMin_{\bar{x}}$. Figure 2 is the original figure that shows how *KKAdd* is computed.

KKMax, uses the d value of the node and adds the minimum f in the other direction, that is, $KKMax_x(n) = d_x(n) + fMin_{\bar{x}}$. Similarly, Figure 3 is the original figure that shows how *KKMax* is computed.

The main disadvantage of this method is the loss of consistency when *KKAdd* or *KKMax* are used as the priority function. Other works have tried to exploit *KKAdd* and *KKMax* in different ways, like computing d in one direction only (Kaindl et al. 1999; Wilt and Ruml 2013) or using it

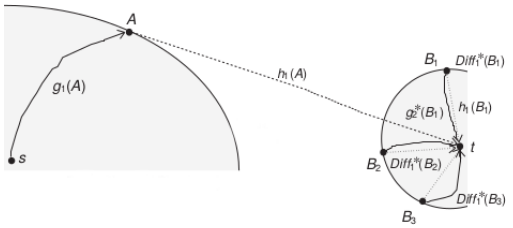

Figure 2: Figure 7 of Kaindl and Kainz (1997).

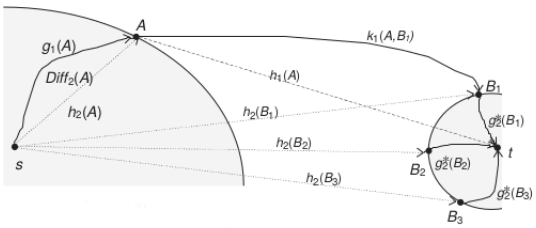

Figure 3: Figure 8 of Kaindl and Kainz (1997).

only to discard nodes after a solution has been found (Auer and Kaindl 2004). Still, this does not exploit all the information derived from heuristic inaccuracies, often requires multiple decisions and/or parameters and is difficult to integrate with other Bi-HS algorithms.

Nevertheless, *KKAdd* and *KKMax* are in fact a lower bound on the cost of any solution going through a given node, and thus can fulfill the same role as the *g bound* and the *f bounds*. In addition, the minimum node values used can be relative, and not absolute like in the original definition, if the purpose is delaying a node at level C. For instance, a forward node n can be delayed if $f_f(n)+dMin_b > C$ — which corresponds to the value of $KKAdd_f(n)$ — even if $dMin_x$ is relative. Likewise, a backward node n' can be delayed if $fMin_f + d_b(n') > C$, which corresponds to $KKMax_f(n')$. This allows us to define two *KK bounds*, one per direction.

Definition 5. *(KK bounds). The forward KK bound is defined as $fMin_f + dMin_b$. The backward KK bound is defined as $fMin_b + dMin_f$.*

Because the *KK bounds* trivially dominate the *f bounds*, the condition in Line 2 of Algorithm 2 can be replaced by:

$$C < max(gMin_f + gMin_b + \epsilon, \\ fMin_f + dMin_b, fMin_b + dMin_f) \quad (1)$$

Interestingly, the delaying rule of the forward *KK bound* corresponds to *KKAdd* for forward nodes and to *KKMax* for backward nodes, and *vice versa* for the backward *KK bound*. Thus, both methods, originally thought to be different techniques, are in fact part of the same lower-bound definition, exemplifying how the lower bound framework can easily integrate Kaindl and Kainz's contribution.

To compute the *KK bounds*, three-dimensional *g-f-d* buckets can be used instead of *g-f* ones. The fixpoint computation of minimum node values can also be modified to account for $dMin_x$ and the delaying rule of the *KK bounds*.

Bidirectional Estimates and BAE*

Before the interest on bidirectional search was rekindled, Sadhukhan proposed an interesting bidirectional best-first algorithm based on accumulated errors along paths, BAE*[1], first at a national conference (Sadhukhan 2012) and later in a journal article (Sadhukhan 2013) with a correction on the latter that drops the need for symmetric heuristics (Sadhukhan 2015). Given node n, BAE* uses a priority function defined as the total accumulated error $TE_x(n) = FE_x(n) + BE_x(n) = (g_x(n) + h_x(n) - h_0) + (g_x(n) - h_{\bar{x}}(n))$, where h_0 is $h_f(start)$ forward and $h_b(goal)$ backward. To prove optimality BAE* uses the lower bound $\frac{1}{2}(h_f(start) + h_b(goal)) + \frac{1}{2}(TEMin_f + TEMin_b)$. Note however that h_0 in direction x is a constant subtracted from all nodes, so one may choose not to subtract it without changing the ranking of the priority function. Such a priority function can be reformulated in terms more similar to the previous ones: $TE_x(n) + h_0 = f_x(n) + d_x(n)$. This can be represented by a single node value, which we define as b.

Definition 6. *(b value of a node). The b value of a node n is defined as $b_x(n) = f_x(n) + d_x(n)$.*

We choose b because it aggregates estimates from both h_f and h_b and so b is a bidirectional estimate. f and d are monotonically increasing along paths and maintain consistency (Hart, Nilsson, and Raphael 1968; Wilt and Ruml 2013), so we can prove as a corollary that b does too. The same result can be derived from Sadhukhan's proofs about the total accumulated error along paths (Sadhukhan 2013).

Corollary 1. *(Consistency of the b value). If h_f and h_b are consistent, b is monotonically increasing along paths, and expanding by minimum b ensures that the g value of a node upon expansion is optimal.*

$bMin_x$ is then the minimum b value among all expandable nodes in $Open_x$. Reformulating BAE*'s global lower bound, we can define a new bound: the *b bound*.

Definition 7. *(b bound). The b bound is defined as $(bMin_f + bMin_b)/2$ rounded up to the next multiple of ι, that is, $\iota\lceil\frac{(bMin_f + bMin_b)/2}{\iota}\rceil$.*

The corresponding delaying rule using the individual *b bound* is $b_x(n) > 2C - bMin_{\bar{x}}$. A proof of the admissibility of the *b bound* can be produced from BAE*'s own proofs, but here we will prove it using Kaindl and Kainz's terms to offer an alternative and hopefully clearer picture on the relationship between BAE* and Kaindl and Kainz's work.

Theorem 2. *(Admissibility of the b bound). If a problem has a solution and $C = C^*$, the b bound will not delay a node belonging to a path of cost C.*

Proof. Let $n \in Open_x$ and $n' \in Open_{\bar{x}}$ be expandable nodes at layer C such that $b_f(n) = bMin_f$ and $b_b(n') = bMin_b$, let P be an optimal path, no solution has been found, and $C = C^*$. There are three possibilities:

[1]We believe that BAE* stands for Bidirectional A* with Error; Sadhukhan never defines the acronym.

- Both nodes belong to the same optimal path, that is, $n, n' \in P$. The sum $b_f(n) + b_b(n')$ can be decomposed as $f_f(n) + d_f(n) + f_b(n') + d_b(n')$. From Corollaries 5.1 and 5.2 of Kaindl and Kainz (1997) we know that $f_f(n) + d_b(n') \leq C^*$ and $f_b(n') + d_f(n) \leq C^*$. Hence, $f_f(n) + d_b(n') + f_b(n') + d_f(n) \leq 2C$, so $b_f(n) + b_b(n') \leq 2C$ and thus $b_f(n) \leq 2C - \mathit{bMin}_b$ and $b_b(n') \leq 2C - \mathit{bMin}_f$, and neither will be delayed.
- $n \in P$ but $n' \notin P$ and there is at least one expandable node n'' such that $n'' \in P$ and $b_x(n'') \geq b_x(n')$. Then the value of the sum $b_f(n) + b_b(n')$ cannot be higher than in the previous case and hence n will not be delayed. The opposite case in which $n' \in P$ but $n \notin P$ is analogous.
- $n \in P$ but $n' \notin P$ and all nodes both in $Open_{\bar{x}}$ and in P have already been delayed by another bound. This cannot happen, as a node on an optimal path cannot be delayed when $C = C^*$ by a bound that maintains admissibility.

□

Having proved this, the *b bound* can be added to the condition in Line 2 of Algorithm 2, making it:

$$\begin{aligned} C < \mathit{max}(&\mathit{gMin}_f + \mathit{gMin}_b + \epsilon, \\ &\mathit{fMin}_f + \mathit{dMin}_b, \mathit{fMin}_b + \mathit{dMin}_f, \\ &\iota\lceil\frac{(\mathit{bMin}_f + \mathit{bMin}_b)/2}{\iota}\rceil) \end{aligned} \tag{2}$$

Informally, the relationship between the *KK bounds* and the *b bound* can be described the following way: given two nodes $n \in \mathit{Open}_x$ and $n' \in \mathit{Open}_{\bar{x}}$, when n tries to use $f_{\bar{x}}(n')$ or $d_{\bar{x}}(n')$ to check if it is delayed by a *KK bound*, the *KK bound* in the opposite direction may delay either node as well using the opposite pair of node values. Since computing all pairwise combinations of *KK bounds* is akin to a front-to-front heuristic and hence *a priori* computationally expensive, the averaged minimum b values are taken instead.

Because b depends on f and d, it can be easily computed using the *g-f-d* buckets used with the *KK bounds*. Nevertheless, explicitly keeping b values can lead to a more efficient implementation, as in our implementation of BAE*. Also, both bMin_x and the delaying rule of the *b bound* can be included in the fixpoint computation of minimum node values.

Comparison of *b* and *KK bounds*

Now we show that the *b bound* and the *KK bounds* do not dominate each other. Figures 4 and 5 prove this. In these pictures, bidirectional arrows represent the heuristic value of both h_f and h_b, and all visible edges have a cost of 1.

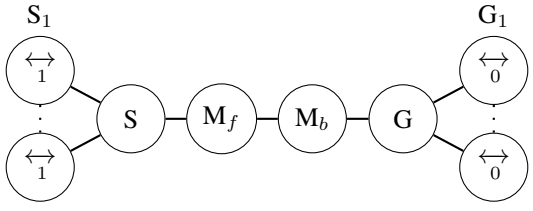

Figure 4: The forward *KK bound* is higher than the *b bound*.

Figure 4 shows how the forward *KK bound* yields a higher value than the *b bound*. Assume that S, G, M_f, and M_b have been expanded, so a solution of cost $U = 3$ has been found. All nodes n in S_1 have $f_f(n) = 2$ and $d_f(n) = 0$; all nodes n' in G_1 have $f_b(n') = 1$ and $d_b(n') = 1$. In this case, $\mathit{fMin}_f = 2, \mathit{dMin}_f = 0, \mathit{bMin}_f = 2, \mathit{fMin}_b = 1, \mathit{dMin}_b = 1$ and $\mathit{bMin}_b = 2$, so the value of the forward *KK bound* is $\mathit{fMin}_f + \mathit{dMin}_b = 3$, and the value of the *b bound* is $(\mathit{bMin}_f + \mathit{bMin}_b)/2 = 2$. The forward *KK bound* proves optimality, as $C = U = 3$, but with only the *b bound* an arbitrary number of additional nodes will have to be expanded.

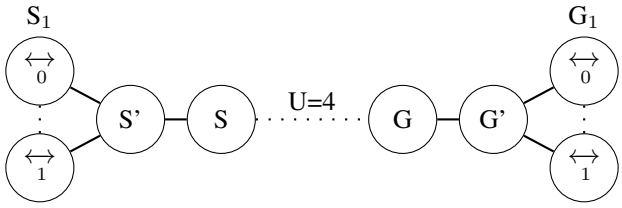

Figure 5: The *b bound* is higher than either *KK bound*.

Figure 5 shows the opposite case. Assume that S, S', G and G' have been expanded, and a solution of cost $U = 4$ has been found. S_1 contains nodes such that $f_f(n) = 3$ and $d_f(n) = 1$ and such that $f_f(n) = 2$ and $d_f(n) = 2$, and similarly G_1 contains nodes such that $f_b(n) = 3$ and $d_b(n) = 1$ and such that $f_b(n) = 2$ and $d_b(n) = 2$. Thus, $\mathit{fMin}_f = 2, \mathit{dMin}_f = 1, \mathit{bMin}_f = 4, \mathit{fMin}_b = 2, \mathit{dMin}_b = 1$ and $\mathit{bMin}_b = 4$, so the value of either *KK bound* is $\mathit{fMin}_x + \mathit{dMin}_{\bar{x}} = 3$, and the value of the *b bound* is $(\mathit{bMin}_f + \mathit{bMin}_b)/2 = 4$. The *b bound* proves optimality, as $C = U = 4$, but if only the *KK bounds* are used an arbitrary number of additional nodes will have to be expanded.

Experiments

In this section we present an exhaustive experimental evaluation of relevant Bi-HS algorithms. All algorithms use ϵ if applicable. The algorithms used are:

- **A* and BS*.** BS* (p) is the original BS* (Kwa 1989); it uses Pohl's cardinality criterion, expanding in the direction of the smallest open list. BS* (a) alternates directions.
- **NBS and NBB.** NBS (Chen et al. 2017) and NBB (Alcázar, Barley, and Riddle 2019), both near-optimal algorithms. NBB is equivalent to NBS_a (Shperberg et al. 2019a) apart from its pseudocode; we choose NBB because its bucket-based implementation is the basis for DBS and DBBS (described below).
- **DVCBS and DVCBS_a.** Developed by Shperberg *et al.* (2019a), the reported state of the art.
- **GBFHS and GBFHS_e.** GBFHS as in Barley *et al.* (2018), but it uses Pohl's cardinality criterion as the split function: when updating its g limits, GBFHS computes the number of expandable nodes on both open lists after increasing the g limit of either direction, and increases the limit that leads to having fewer expandable nodes. Initially the f limit is set to $max(h_f(\mathit{start}), h_b(\mathit{goal}))$ and both g limits are set to 0. All limits are increased by ι and not by 1.

Algorithm	GAP-0			GAP-1			GAP-2			GAP-3			GAP-4			GAP-5			GAP-6		
NBS	174	71	0.22	7310	7169	0.96	150k	149k	0.98	786k	786k	1	1448k	1448k	1	1624k	1624k	0.98	1626k	1624k	0.82
NBB	469	61	0.12	5474	5217	0.92	115k	111k	0.96	620k	617k	0.98	1346k	1345k	0.94	1598k	1598k	0.84	1626k	1622k	0.54
DBS (a)	521	58	0.06	23724	3308	0.06	163k	54k	0.14	425k	289k	0.20	549k	510k	0.28	665k	628k	0.30	**786k**	717k	0.28
DBBS (a)	521	58	0.06	13924	1780	0.12	136k	40k	0.16	404k	240k	0.22	544k	486k	0.28	664k	617k	0.30	**786k**	711k	0.28
A*	**72**	45	0	8431	8393	0	352k	348k	0	6865k	6827k	0	-	-	-	-	-	-	-	-	-
BS* (a)	379	96	0	10712	10868	0.02	443k	432k	0	9462k	9667k	0	-	-	-	-	-	-	-	-	-
BS* (p)	220	62	0	6148	6061	0.02	246k	246k	0	5452k	5406k	0	-	-	-	-	-	-	-	-	-
DVCBS	161	45	0.06	5798	5703	0.86	94k	93k	0.96	482k	479k	0.70	937k	899k	0.20	1130k	1039k	0.02	1168k	1053k	0
$DVCBS_a$	365	46	0	10681	4595	0.44	122k	83k	0.36	630k	452k	0.14	1060k	897k	0	1162k	1039k	0	1238k	1053k	0
GBFHS	988	**42**	0	58126	5190	0	717k	116k	0	1686k	632k	0	1080k	914k	0	1066k	1039k	0	1064k	1053k	0
$GBFHS_e$	112	44	0	5226	5007	0	105k	104k	0	584k	581k	0	923k	913k	0	1063k	1039k	0	1081k	1053k	0
BAE* (a)	90	65	0	**654**	**552**	0	**15k**	**11k**	0	**132k**	**113k**	0	**469k**	465k	0	1107k	1091k	0	1793k	1775k	0
BAE* (p)	88	64	0	729	620	0	18k	13k	0	154k	135k	0	481k	476k	0	1106k	1090k	0	1785k	1768k	0
DBS (p)	293	45	0.02	9027	2672	0.28	81k	46k	0.24	317k	250k	0.30	512k	464k	0.30	667k	617k	0.24	794k	718k	0.28
DBBS (p)	293	45	0.02	4110	902	0.22	34k	21k	0.48	230k	170k	0.44	474k	**430k**	0.36	**649k**	**587k**	0.36	791k	**702k**	0.22

Table 1: Pancakes. Dashed entries mean that there were instances in which the solver ran out of memory.

GBFHS can behave badly in the last layer, as it expands nodes such that $f(n) = C^*$ before increasing the g limits, delaying the collision. $GBFHS_e$ (eager GBFHS) increases the g limits as soon as the f limit is increased except in the initial step, losing some information when deciding which g limit to increase but allowing early collisions.

- **DBS and DBBS.** DBS (d-using Bidirectional Search) uses *g-f-d* buckets and the g and *KK bounds* to delay nodes and prove optimality, and performs a fixpoint computation of all minimum node values. DBBS (d and b-using Bidirectional Search) also uses the *b bound*. Both expand by minimum g and break ties by minimum f, and additionally among d buckets, by minimum d. We choose this because the most successful algorithms so far (NBB/NBS_a, DVCBS and GBFHS) do so as well, and because we want to display how using additional bounds increasingly improves the performance of NBB. However, other criteria may be better both to raise C faster and to have a better last-layer tie-break. DBS (a) and DBBS (a) expand alternating directions like NBB, and thus retain near-optimality. DBS (p) and DBBS (p) use Pohl's cardinality criterion, but count only expandable nodes with minimum g, similar to DVCBS and GBFHS.
- **BAE*.** Same as BS*, but the priority function uses $b_x(n)$ instead and the termination criterion is the *b bound*.

Experiments show average total expanded nodes, average necessarily-expanded nodes (expanded when $C < C^*$) and the ratio of problems with no expansions in the last layer, that is, an optimal solution was found before C reached C^*. For nodes, best results are in bold. NBS, NBB, DBS (a) and DBBS (a) are listed first, as they are near-optimal.

Table 1 shows results for 100 random instances in the **14-Pancake Puzzle** with the GAP heuristic (Helmert 2010). The first k pancakes of the target state are ignored to get an asymmetric weaker heuristic. In terms of necessarily-expanded nodes, DBS (a) is clearly stronger than NBB, the difference increasing as the heuristic degrades. DBBS expands fewer nodes than DBS, especially the (p) version for heuristics of intermediate strength. BAE* is very competitive for heuristics of intermediate strength, but the advantage of using the *b bound* decreases relative to the *g bound* for weak heuristics, and becomes the worst algorithm among the modern Bi-HS ones for GAP-6. DBS and DBBS keep a robust behavior thanks to their combination of bounds, and become the best algorithms for GAP-5 and GAP-6.

We also made the heuristic symmetric by abstracting the first k pancakes away using a relative-order abstraction (Helmert and Röger 2010) and then computing GAP. We omit the results for lack of space, but DBBS in this case notably outperforms BAE* even for heuristics of intermediate strength. We hypothesize that, because the heuristic is symmetric and the search space is largely so as well, the best splits for all but the strongest heuristics will occur around the middle, in which case the *g bound* is most useful.

Algorithm	ToH-12 (10+2)			ToH-12 (8+4)			ToH-12 (6+6)		
NBS	233k	233k	1	654k	653k	0.94	682k	663k	0.56
NBB	230k	230k	0.98	645k	645k	0.90	678k	662k	0.56
DBS (a)	140k	118k	0.38	531k	488k	0.12	640k	606k	0.14
DBBS (a)	70k	66k	0.74	307k	286k	0.52	621k	583k	0.20
A*	276k	276k	0	1926k	1925k	0	3268k	3239k	0
BS* (a)	175k	177k	0	1180k	1161k	0	1603k	1599k	0
BS* (p)	230k	235k	0	929k	918k	0	2004k	1988k	0
DVCBS	224k	224k	0.94	613k	601k	0.32	664k	636k	0.16
$DVCBS_a$	219k	217k	0.74	628k	601k	0.06	661k	628k	0.08
GBFHS	207k	162k	0	1246k	1071k	0	1524k	1384k	0
$GBFHS_e$	207k	206k	0	621k	615k	0	653k	627k	0
BAE* (a)	47k	46k	0	187k	186k	0	383k	382k	0
BAE* (p)	**46k**	**45k**	0	**185k**	**182k**	0	**375k**	**374k**	0
DBS (p)	113k	105k	0.70	463k	443k	0.58	601k	570k	0.24
DBBS (p)	58k	54k	0.80	232k	221k	0.66	496k	479k	0.58

Table 2: Towers of Hanoi

Table 2 shows results for 50 instances of the **12-disk 4-peg Towers of Hanoi** problem with (10+2), (8+4) and (6+6) additive PDBs (Felner, Korf, and Hanan 2004). The *KK bounds* are very useful *e.g.* DBS (a) expands almost half of NBB's nodes with (10+2). The *b bound* is even stronger, making DBBS and BAE* the most efficient algo-

rithms. BAE* has an edge over DBBS because it expands nodes by minimum b instead of minimum g, which raises C faster in this domain. As the heuristic degrades, DBS' advantage decreases, but DBBS and BAE* retain a substantial margin. The *b bound* seems to be more useful than the *KK bounds* because additive heuristics compensate for their weaknesses and avoid extreme $fMin_x$ and $dMin_x$ values, probably benefiting the averaging behavior of the *b bound*. Still, a deeper analysis is necessary to better understand this.

Algorithm	15 Puzzle		
NBS	12748k	12710k	0.93
NBB	12067k	11739k	0.86
DBS (a)	26590k	10736k	0.07
DBBS (a)	22184k	10159k	0.19
A*	15550k	14700k	0
BS* (a)	19355k	19305k	0
BS* (p)	12001k	11942k	0
DVCBS	11670k	11590k	0.78
DVCBS$_a$	11934k	10660k	0.55
GBFHS	42532k	10645k	0
GBFHS$_e$	10753k	10645k	0
BAE* (a)	**2707k**	**2700k**	0
BAE* (p)	2837k	2829k	0
DBS (p)	10187k	8011k	0.66
DBBS (p)	3915k	3059k	0.69

Table 3: Sliding Tile Puzzle

Table 3 shows results for 100 instances of the **15 Sliding Tile Puzzle** (Korf 1985) with Manhattan Distance. DBS (p) and DBBS (p) are much better than their alternating counterparts, which implies that the new bounds are useful in finding good splits too. Expanding by b like BAE* is very advantageous, but DBBS (p) is close despite expanding by g. DBS and DBBS expand many more total nodes than necessarily-expanded nodes due to delayed nodes along optimal paths.

Algorithm	Mazes			DAO		
NBS	84011	84006	0.86	6676	6561	0.37
NBB	84007	83993	0.86	6876	6543	0.35
DBS (a)	78633	78563	0.52	6200	5866	0.21
DBBS (a)	78525	78458	0.53	6191	5858	0.21
A*	99396	99369	0	5406	5321	0
BS* (a)	100330	99369	0	6609	6567	0
BS* (p)	87726	87760	0	6200	6132	0
DVCBS	93543	93434	0.42	5545	5157	0.16
DVCBS$_a$	93145	93032	0.43	5545	5152	0.17
GBFHS	98573	98547	0.18	**5225**	**5030**	0.03
GBFHS$_e$	78212	78206	0.93	6262	6088	0.24
BAE* (a)	80835	80809	0	6718	6668	0
BAE* (p)	**74069**	**74033**	0	5995	5861	0
DBS (p)	75167	75131	0.75	5600	5318	0.28
DBBS (p)	75533	75501	0.76	5595	5314	0.28

Table 4: Grids

Table 4 shows results for **grid-based pathfinding** benchmarks (Sturtevant 2012) with the octile heuristic, using mazes and Dragon Age: Origins (DAO) maps. Diagonal moves are allowed with a cost of 1.5, so costs are non-unitary. Mazes are built to misguide the heuristic, and indeed DBS, DBBS and BAE* are able to leverage this. DVCBS and GBFHS are negatively affected, with only GBFHS$_e$ able to find good splits. DAO maps are more conventional, and using heuristic inaccuracies is not much better than just using A*, although DBS and DBBS are still better than NBB.

In terms of nodes per second, using more dimensions in the open list and doing a fixpoint computation can affect performance. With a small range of node values (all unitary-cost domains here), DBS and DBBS are not substantially slower. The opposite is true in Mazes and DAO, with a large range of node values. Here the algorithms that use a heap (A*, BS*, BAE*, NBS) are the fastest; those that use bi-dimensional buckets (NBB, DVCBS, GBFHS) are up to 1 order of magnitude slower; those with three-dimensional buckets and fixpoint computation (DBS and DBBS) are up to 2 orders of magnitude slower. A preliminary analysis shows that the fixpoint computation ends on average after a few steps. However, a high range of node values means that the number of buckets is high (so iterating through them is costlier) and the buckets are smaller (so minimum node values are recomputed often). Nevertheless, results may be different when the bottleneck is the heuristic or the successor generation, like in automated planning. Also, more efficient implementations should be investigated and additional experimentation should find which bounds are worth using.

Regarding total nodes, the new bounds often delay nodes on optimal paths *e.g.* in Pancake GAP-1, NBB explores the last layer only 8% of the time, but DBS (a) does it 94% of the time, with catastrophic results in terms of total nodes. GBFHS and DVCBS$_a$ display sometimes a very bad last-layer behavior too. Hence, implementing *RunTieBreaker()* is an interesting alternative for these algorithms.

Overall, BAE* stands out as the most efficient algorithm in the presented data. However, this does not mean that BAE* will always expand fewer nodes, as other priority functions may yield better results. For example, DBBS expanding by b and DBS expanding by f are often similar or better than BAE* *e.g.* in Sliding Tile Puzzle, the best domain for BAE*, DBS (a) by f expands 1806k necessarily-expanded nodes, and DBBS (a) by b, 1701k, compared to BAE*'s 2700k. Time can also be improved by using fewer node values *e.g.* just g and b values. Unfortunately, both questions are out of scope here, and belong to future work.

Conclusions and Future Work

In this paper we have defined individual bounds within the lower-bound framework of bidirectional search. These bounds seamlessly integrate both modern techniques and techniques that for years remained hard to exploit. They generalize over Kaindl and Kainz's and Sadhukhan's works and are easy to understand and to exploit in modern Bi-HS algorithms. Results show a great leap in performance, breaking the limits imposed by near-optimality thanks to the use of heuristic inaccuracies and pushing the state of the art. Due to this and to how relatively simple it is to implement algorithms using the newer bounds, we personally argue that, if backward search is possible and consistent heuristics are

available, front-to-end bidirectional search should be tried first over unidirectional algorithms like A*.

Two main lines of research are now open, one theoretical and one empirical: first, the must-expand pairs defined by Eckerle *et al.* (2017) and its derived definitions, like near-optimality, must be modified to work with individual bounds. Second, a thorough empirical analysis is necessary to understand when and why the different bounds are useful, which criteria to use to expand nodes (*e.g.* if the forward *KK bound* is the one raising C, expand by f forward and by d backward), how the last-layer behavior is different from the C-rising process, and how to modify newer algorithms (like NBS, DVCBS and GBFHS, arguably more complex than DBS and DBBS) to exploit the individual bounds.

References

Alcázar, V.; Barley, M.; and Riddle, P. J. 2019. A theoretical comparison of the bounds of MM, NBS, and GBFHS. In *Proceedings of the Twelfth International Symposium on Combinatorial Search, SoCS*, 160–161.

Auer, A., and Kaindl, H. 2004. A case study of revisiting best-first vs. depth-first search. In *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, 141–145.

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

Barley, M. W.; Riddle, P. J.; Linares López, C.; Dobson, S.; and Pohl, I. 2018. GBFHS: A generalized breadth-first heuristic search algorithm. In *Proceedings of the Eleventh International Symposium on Combinatorial Search, SoCS*, 28–36.

Burns, E. A.; Hatem, M.; Leighton, M. J.; and Ruml, W. 2012. Implementing fast heuristic search code. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SoCS*, 25–32.

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

Eckerle, J.; Chen, J.; Sturtevant, N. R.; Zilles, S.; and Holte, R. C. 2017. Sufficient conditions for node expansion in bidirectional heuristic search. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS*, 79–87.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *JAIR Journal of Artificial Intelligence Research* 22:279–318.

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

Helmert, M., and Röger, G. 2010. Relative-order abstractions for the pancake problem. In *ECAI European Conference on Artificial Intelligence*, 745–750.

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

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

Kaindl, H., and Kainz, G. 1997. Bidirectional heuristic search reconsidered. *J. Artif. Intell. Res.* 7:283–317.

Kaindl, H.; Kainz, G.; Steiner, R.; Auer, A.; and Radda, K. 1999. Switching from bidirectional to unidirectional search. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI*, 1178–1183.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artif. Intell.* 27(1):97–109.

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

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

Sadhukhan, S. K. 2012. A new approach to bidirectional heuristic search using error functions. In *1st International Conference on Intelligent Infrastructure at the 47th Annual National Convention Computer Society of India (CSI-2012)*, 239–243.

Sadhukhan, S. K. 2013. Bidirectional heuristic search using error estimate. *CSI Journal of Computing* 2(1-2):S1:57–S1:64.

Sadhukhan, S. K. 2015. Letter to the editor - corrections. *CSI Journal of Computing* 2(4):80.

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 *Proceedings of the Tenth International Symposium on Combinatorial Search, SoCS*, 82–90.

Shaham, E.; Felner, A.; Sturtevant, N. R.; and Rosenschein, J. S. 2018. Minimizing node expansions in bidirectional search with consistent heuristics. In *Proceedings of the Eleventh International Symposium on Combinatorial Search, SoCS*, 81–98.

Shperberg, S.; Felner, A.; Sturtevant, N. R.; Hayoun, A.; and Shimony, E. S. 2019a. Enriching non-parametric bidirectional search algorithms. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, 2379–2386.

Shperberg, S. S.; Felner, A.; Sturtevant, N. R.; Shimony, S. E.; and Hayoun, A. 2019b. Improving bidirectional heuristic search by bound propagation. *Proceedings of the Twelfth International Symposium on Combinatorial Search, SoCS* 106–114.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE Trans. Comput. Intellig. and AI in Games* 4(2):144–148.

Wilt, C. M., and Ruml, W. 2013. Robust bidirectional search via heuristic improvement. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 954–961.