Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Limitations of Front-to-End Bidirectional Heuristic Search

Joseph K Barker and **Richard E Korf**
{jbarker,korf}@cs.ucla.edu
4732 Boelter Hall
Los Angeles, CA 90095

Abstract

We present an intuitive explanation for the limited effectiveness of front-to-end bidirectional heuristic search, supported with extensive evidence from many commonly-studied domains. While previous work has proved the limitations of specific algorithms, we show that *any* front-to-end bidirectional heuristic search algorithm will likely be dominated by unidirectional heuristic search or bidirectional brute-force search. We also demonstrate a pathological case where bidirectional heuristic search *is* the dominant algorithm, so a stronger claim cannot be made. Finally, we show that on the four-peg Towers Of Hanoi with arbitrary start and goal states, bidirectional brute-force search outperforms unidirectional heuristic search using pattern-database heuristics.

1 Introduction

Bidirectional search is a well-known algorithm schema for pathfinding problems. Two searches are simultaneously done forward from the start state and backward from the goal. A solution is found when the two search frontiers intersect, although more search may be required to prove optimality. Bidirectional brute-force search can be an effective search technique. In a unidirectional brute-force search of a space with unit edge costs, branching factor b, and a solution depth d, the number of nodes generated is $O(b^d)$. In a bidirectional search that meets at the midpoint at depth $d/2$, however, the number of nodes generated is only $O(b^{d/2})$.

A refinement is to guide search with a heuristic. In a *front-to-end* search, the heuristic is used to estimate cost to only the start or goal state. The earliest such algorithm is the Bidirectional Heuristic Path Algorithm (BHPA) (Pohl 1971), Other variants include BS* (Kwa 1989). Despite initially-promising results, these algorithms are not widely used.

We present an explanation for why front-to-end bidirectional heuristic search is generally ineffective. However, we also give a counterexample of a graph where bidirectional heuristic search *does* outperform both unidirectional heuristic and bidirectional brute-force search, and so there is no general proof of its ineffectiveness. Our theory gives a strong intuition for why the technique is usually not helpful, despite such pathological cases, and is supported by extensive evidence from many well-studied search domains. While there exist cursory discussions of similar explanations, we are unaware of a previous thorough treatment in the literature.

In our experiments, we examine the four-peg Towers Of Hanoi problem with arbitrary start and goal states, where we find that bidirectional brute-force search outperforms unidirectional heuristic search. We also find that, contrary to a common belief, bidirectional heuristic searches with strong heuristics tend to find optimal solutions late in search.

We consider only domains with unit and additive g costs, which are most domains commonly studied in the literature.

2 Background

2.1 Front-To-End Bidirectional Search

A bidirectional search does simultaneous forward and reverse searches. The reverse search is done from the goal to the start, using the reverse of the standard operators. In a problem space without invertible operators, the reverse operators for a state s are those which generate s when applied to some other state. Both search directions have a frontier of the deepest nodes generated so far. In an algorithm like depth-first iterative deepening (Korf 1985) this frontier may not be explicitly represented. Any time the frontiers intersect—meaning the same node has been generated in both directions—a path has been found from the start to the goal. The first path found is not necessarily the cheapest, so search continues until a solution is proven optimal.

Both directions of search can be explored using a heuristic. The earliest bidirectional heuristic search algorithm is BHPA (Pohl 1971), which does A* searches in both directions. When the frontiers intersect, a solution is found and search continues until a solution has been proven optimal. Search can stop under two conditions: either the minimum f cost in either frontier, or the sum of the minimum g costs of both frontiers, is at least the cost of the best solution found so far. In the first case, if all nodes on one frontier have f cost at least that of the cost of the current best solution, then—assuming an admissible heuristic—any additional paths found through those nodes cannot be lower than the current best solution cost (Kaindl and Kainz 1997). The second case is inherited from Djikstra's algorithm: g cost cannot decrease with depth and any additional solutions must pass through a node on each frontier, so their cost must be at least the sum of the minimum g costs on both frontiers.

When using a consistent heuristic, performance can be improved by not expanding a node if it has already been expanded in the opposite direction (Kwa 1989). Consistency guarantees that the g cost of an expanded node corresponds to its lowest-cost path from the root. If the same node is expanded in both directions, the path generated is the concatenation two optimal paths to that node, and no cheaper path can be found through that node. The descendants of that frontier intersection (in either direction) need not be generated. This is the primary improvement of BS* over BHPA.

This paper considers *front-to-end* search specifically, unless explicitly stated otherwise. A front-to-end heuristic evaluation is done from a node to either the start or goal node, depending on the direction of search. By contrast, a *front-to-front* heuristic evaluation is done from a node to every node on the opposing frontier and the minimum is used, giving more accurate values at the expense of many more heuristic evaluations. While such techniques are beyond the scope of our analysis, we briefly discuss them in section 3.3.

2.2 Kaindl and Kainz 1997

Kaindl and Kainz (1997) performed one of the first analyses of bidirectional heuristic search, focusing specifically on BHPA. They showed that the then-accepted explanation for its ineffectiveness—that the two search frontiers pass by each other without intersecting—was incorrect. They proved that the best-case performance of BHPA is only slightly better than the best-case performance of unidirectional A*. In fact, the only improvement of BHPA over A* is that it may explore fewer nodes with f cost *equal* to the optimal solution cost C^*. However, the total number of nodes explored in both directions with f cost *less than* C^* will be least as large as the number explored by unidirectional A*.

They analyzed BHPA specifically, which allows the two search frontiers to pass through each other. They explicitly exclude algorithms that do not search past frontier intersections when using a consistent heuristic. If the number of nodes whose generation is prevented is large enough, a bidirectional heuristic search algorithm may yet be effective.

They also found that bidirectional heuristic search algorithms find optimal solutions early, and spend most of their time proving solution optimality. As we show in section 4.3, however, with strong heuristics this is *not* true: optimal solutions tend to be found quite late in search.

2.3 Edelkamp and Schrödl 2011

In *Heuristic Search* (2011), Edelkamp and Schrödl hypothesize that bidirectional heuristic search perfoms poorly because the mean depth of nodes generated in heuristic search tends to be near the solution midpoint. They claim that if the search frontiers meet near this midpoint then bidirectional heuristic search stores roughly twice as many nodes on its open lists as unidirectional heuristic search, but that a bidirectional search may perform better if the frontiers intersect before or after the midpoint. They illustrate their hypothesis with a graph of nodes generated at each search depth in one 15-puzzle problem instance with an optimal solution depth of 15 (the average solution depth is 53) using the Manhattan-distance heuristic. The distribution of these nodes is indeed roughly centered on the solution midpoint.

As we show in section 4.1, however, the premise of this hypothesis is incorrect: unidirectional searches with strong heuristics tend to expand the vast majority of nodes *before* the solution midpoint. Furthermore, we show that regardless of the mean g cost, a bidirectional heuristic search will generally be outperformed by either a unidirectional heuristic or bidirectional brute-force search. Our argument is supported with extensive evidence from a large number of domains.

3 Analysis

3.1 Our Explanatory Theory

If a unidirectional heuristic search expands the majority of its nodes deeper than the solution midpoint, we call its heuristic *weak* and show that a bidirectional heuristic search expands no fewer nodes than a bidirectional *brute-force* search. If the majority are expanded at shallower depth than the solution midpoint, then we call it a *strong* heuristic and show that a bidirectional heuristic search would expand *more* nodes than a unidirectional heuristic search.

A balanced bidirectional brute-force search expands no nodes deeper than the solution midpoint. In a bidirectional brute-force search that balances the number of nodes expanded in each direction, the forward and reverse directions intersect at the solution midpoint and nodes deeper than this radius are not expanded. In an unbalanced search, so long as one direction of search does not expand significantly more nodes than the other, search will meet near the solution midpoint will not expand any deep nodes.

Figure 1 shows the number of nodes expanded at each depth in unidirectional and bidirectional brute-force searches of the 16-disk four-peg Towers Of Hanoi where all disks start and end on a single peg. The dark gray nodes are expanded in a bidirectional search, and the additional light gray nodes are expanded in a unidirectional search, all of which are deeper than the solution midpoint.

Adding a *weak* heuristic to a bidirectional brute-force search cannot prevent it from expanding additional nodes. In an admissible heuristic search, nodes with f cost greater than C^* are not expanded, either by being explicitly pruned (as in IDA* (Korf 1985)) or by being queued for expansion after a solution has been proven optimal (as in A* (Hart, Nilsson, and Raphael 1968)). The f cost of a node n is the sum of two values: the g cost, which is the cost of the current path from the start to n, and the h cost, which is the heuristic estimate from n to the goal. These values are independent, and so nodes with higher g cost tend to have higher f costs. With a consistent heuristic, in fact, f costs are monotonically non-decreasing with increasing g cost (Pearl 1984). As a result, nodes whose expansion is prevented by a heuristic—those with f cost greater than C^*—tend to have high g cost. With a sufficiently weak heuristic, the only nodes whose expansion is prevented have depth greater than the solution midpoint, none of which are expanded in a bidirectional brute-force search, either.

Figure 1 shows increasingly strong unidirectional heuristic searches on a Towers Of Hanoi instance, drawn with

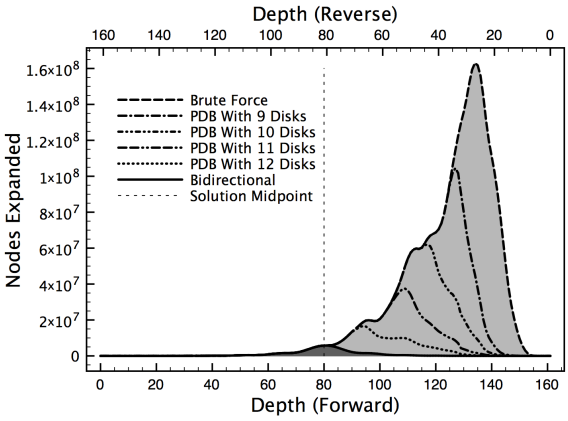

Figure 1: Depths of nodes expanded in searches of a Towers of Hanoi instance. Dashed lines are unidirectional searches with heuristics of varying strengths. Nodes expanded in a unidirectional brute-force search are light gray. Nodes expanded in a bidirectional brute-force search are dark gray.

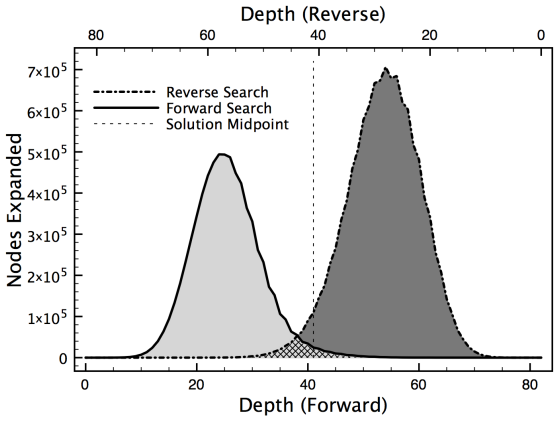

Figure 2: Depths of nodes expanded in forward and reverse searches on a 24-Puzzle instance.

dashed lines. The algorithm used is breadth-first heuristic search (BFHS) (Zhou and Hansen 2006), which expands nodes in breadth-first order while maintaining a global cost cutoff. Any node whose f cost exceeds that cutoff is pruned. We set the cutoff to the optimal solution cost. We used increasingly large pattern databases (PDBs) (Culberson and Schaeffer 1996) to strengthen the heuristic. As expected, fewer nodes are expanded with successively stronger heuristics, and the nodes not expanded are the deepest. The nodes whose expansion is prevented by these weak heuristics are not expanded in the bidirectional brute-force search, either.

With a *strong* heuristic, a bidirectional heuristic search expands more nodes than a unidirectional heuristic search. The majority of nodes expanded in a forward or reverse heuristic search with a sufficiently strong heuristic will be shallower than the solution midpoint. Nodes that are shallower than the midpoint in a reverse search are deeper than the midpoint in a forward search, so combining the two searches to form a bidirectional search expands more nodes than doing either independently.

Figure 2 shows the depth of nodes expanded by heuristic searches of a 24-Puzzle instance using PDBs described in (Korf and Felner 2002). The light gray nodes are expanded in a forward search and the dark gray nodes are expanded in a reverse search; the hashed region is expanded in both. A bidirectional heuristic search expands both regions, while a unidirectional heuristic search expands only one.

With a heuristic that is neither strong nor weak, approximately half of all nodes expanded in a unidirectional heuristic search are shallower than the midpoint. A bidirectional heuristic search that meets at the midpoint allows the forward direction of search to avoid expanding any nodes past the midpoint. However, it also expands nodes in the reverse direction to reach the midpoint. The number of additional nodes expanded in the reverse direction is approximately equal to the number of nodes pruned in the forward direction, and so a bidirectional heuristic search expands no fewer nodes than a unidirectional heuristic search.

In a non-breadth-first search ordering, both directions of search may expand some nodes deeper than the midpoint before they expand shallower nodes. Section 3.2 shows how this can result in pathological cases where a bidirectional heuristic search outperforms the other two algorithms, and discusses why such cases are unlikely. In general, bidirectional heuristic search is no better than the stronger of unidirectional heuristic or bidirectional brute-force search done individually, and may be worse.

3.2 A Pathological Case

This does not prove that bidirectional heuristic search can never be effective. As a counterexample, figure 3 is a pathological graph where bidirectional heuristic search outperforms both unidirectional heuristic and bidirectional brute-force search. Each node in the graph is labeled with its f, g, and h costs, and the graph is entirely symmetrical. It has unit edge costs and, as the h cost of siblings never differs by more than one, the heuristic is consistent. A start-to-goal unidirectional heuristic search expands all nodes with f cost less than the optimal solution cost of six, of which there are 13: all but k, l, and the goal. A bidirectional brute force search expands all nodes up to depth two in both directions, at which point an intersection would be generated at depth three and proved optimal. There are 14 such nodes: all except c and h.

Bidirectional heuristic search expands fewer nodes if it explores children in left-to-right order and breaks f cost ties in favor of lower h cost. Node e will be expanded in the forward direction before the reverse and node f in the reverse direction before the forward; each direction of search will not expand the children of these nodes because their f cost (seven) exceeds the optimal solution cost of six. When node e is considered for expansion in the reverse direction and node f in the forward direction, both can be discarded as they have already been expanded in the opposite direction. Subgraphs A and B will not be expanded and search will expand only 12 nodes: all except k, l, m, and n. Thus, bidirectional heuristic search outperforms unidirectional heuristic and bidirectional brute-force search on this graph.

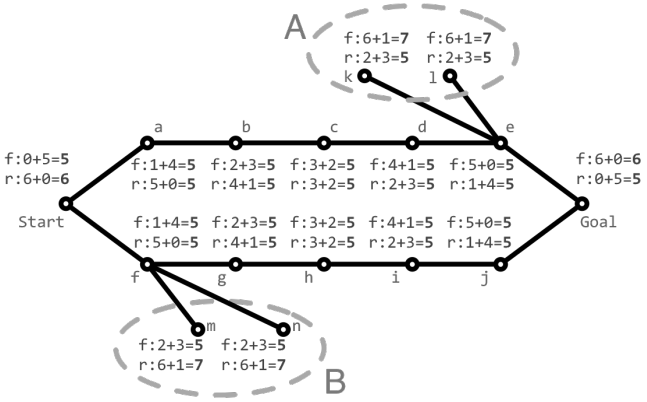

Figure 3: A pathological graph where bidirectional heuristic search outperforms both unidirectional heuristic and bidirectional brute-force search. Nodes are labeled with g-cost + h-cost = f-cost, in the forward (f) and reverse (r) direction.

The size of subgraphs A and B can increased arbitrarily by increasing their depth and branching factor, and their nodes' h costs can be defined so that their f cost in only one direction is less than C^*. With a sufficiently deep optimal solution, both bidirectional brute force and unidirectional heuristic search will expand at least one of A or B, in addition to the middle paths. Bidirectional heuristic search will expand neither, and can thus expand arbitrarily fewer nodes. This counterexample shows that no formal proof of the ineffectiveness of bidirectional heuristic search is likely.

Such a pathological case is unlikely to occur in a real search problem, however. In general, f costs of nodes in a search space increase with depth, so we expect shallower regions to be expanded earlier. This is particularly true with a consistent heuristic, which results in f costs that are monotonically non-decreasing with increased depth. As such, neither direction of search can intersect the other before most nodes with lower depth have been expanded, and a pathological case like that of figure 3 would be very unlikely.

3.3 Caveats

Here we identify some caveats to this theory. We consider only static heuristic functions, but a bidirectional search can use the frontiers to increase heuristic accuracy. Examples of algorithms that do so are KKAdd (Kaindl and Kainz 1997), perimeter search (Dillenburg and Nelson 1994; Manzini 1995), and single-frontier bidirectional search (SFBDS) (Felner et al. 2010). These algorithms all use some variation of front-to-front evaluation to improve heuristic accuracy. The state of the art solver for Peg Solitaire, a domain we examine later, uses the search frontiers to tighten domain-specific pruning constraints (Barker and Korf 2012).

As previously mentioned, even a front-to-end algorithm *can* reduce the number of nodes expanded with f cost equal to C^*. This may mean that, in some domains, a bidirectional heuristic search could have reliably better tie-breaking properties than a unidirectional heuristic search.

Finally, we have so far assumed that the difficulty of a forward and reverse search is roughly equivalent; however, in some domains one direction may be significantly cheaper than the other. If the same direction is consistently significantly easier, then it will be cheaper to do a unidirectional search in that direction, regardless of the weakness of the heuristic. Conversely, if the cheaper direction is unpredictable, then even with a strong heuristic a bidirectional search can be effective as it will solve the majority of the instances in the cheaper direction. This is effectively equivalent to doing a forward and reverse search in parallel and terminating when the first one finds a solution.

4 Experiments

4.1 Empirical Studies of Search Spaces

We substantiated our theory by finding the depths of all nodes expanded in many instances of several domains. We computed the fraction of those nodes that are shallower than the solution midpoint. We expect a unidirectional heuristic search to dominate if this is the majority, and a bidirectional brute-force search to dominate otherwise.

We tested seven domains using state-of-the-art heuristics: the 15 Puzzle, the 24 Puzzle, the Pancake Problem, Peg Solitaire, Rubik's Cube, Top Spin, and the four-peg Towers Of Hanoi. We solved 100 random 15-Puzzle instances using PDBs described in (Korf and Felner 2002). We solved the 50 24-Puzzle instances from (Korf and Felner 2002) using IDA* with PDBs from the same paper. We solved 25 random Rubik's Cube instances using IDA* with an eight-corner-cubie PDB and a nine-edge-cubie PDB using six rotational lookups (Felner et al. 2011). We solved 20 random Top Spin instances with 21 tokens and a four-token tray, using IDA* with three seven-token, additive PDBs and three lookups, as described in (Yang et al. 2008). We solved 60 Pancake Problem instances with 80 pancakes, using IDA* and the gap heuristic (Helmert 2010). We solved all 35 Peg Solitaire instances on the English, French, and Diamond(5) boards using BFHS and heuristics from (Barker and Korf 2012). For the four-peg Towers Of Hanoi, we solved 100 random 20-disk instances with arbitrary start and goal states, using disk-based BFHS and the PDB techniques used in (Korf 2008). This domain is further discussed in section 4.2.

Table 1 summarizes our experiments. Each column shows the fraction of nodes expanded shallower than the midpoint; the first is the mean fraction among all instances, and the last two are the value for the instances with the minimum and maximum fractions. In our IDA* searches, we only consider the last iteration of search (the most expensive) to avoid multiple counting of nodes generated on multiple iterations.

In Peg Solitaire and Towers Of Hanoi, a mean of 6% and 31% of the nodes expanded have shallower depth than the midpoint, respectively; as such, we expect a bidirectional brute-force search to outperform a unidirectional heuristic search. As we will show in section 4.2, bidirectional brute-force search outperforms unidirectional heuristic search on the four-peg Towers Of Hanoi. The state-of-the-art solver for Peg Solitaire is in fact a bidirectional heuristic search algorithm (Barker and Korf 2012). We found that forward search in Peg Solitaire is consistently significantly cheaper than re-

Domain	Mean	Min	Max
15 Puzzle	0.93	0.63	0.99
24 Puzzle	0.96	0.80	1.00
Pancake Problem	0.57	0.49	0.66
Peg Solitaire	0.06	0.02	0.14
Rubik's Cube	0.90	0.84	0.98
Top Spin	0.98	0.86	1.00
Towers of Hanoi	0.31	0.13	0.59

Table 1: Fraction of nodes expanded with with depth less than or equal to the solution midpoint for many instances of several search domains.

verse search due to an imbalance of branching factors so, as discussed in section 3.3, we would expect a unidirectional heuristic search to dominate. Barker and Korf's algorithm is based on BFIDA* (Zhou and Hansen 2006), however, which has worst-case tie-breaking and generates a very large number of nodes with f cost equal to C^*. Their bidirectional approach allows the solver to eliminate the generation of all such nodes, which is the possible contribution of bidirectional heuristic search identified by Kaindl and Kainz. As such, Peg Solitaire is an example of a domain where the exceptions identified in section 3.3 make bidirectional heuristic search an effective algorithm.

In the remaining domains the majority of nodes expanded are shallower than the midpoint, so we expect unidirectional heuristic search to dominate. Indeed, we are unaware of a front-to-end bidirectional search in the literature that outperforms the best unidirectional search on these domains[1]

As an interesting case, the gap heuristic for the Pancake Problem is extremely strong: in all instances we tested very few nodes expanded had f cost less than C^*; often *none* did. With a *perfect* heuristic a search algorithm expands only the nodes on a single optimal solution path, and the mean g cost of expanded nodes is $C^*/2$. With a strong heuristic, we predict that most nodes are expanded with g cost less than $C^*/2$; the gap heuristic is close enough to perfect, though, that the g cost distribution of nodes expanded is relatively flat and the mean is closer to the solution midpoint.

As the absence of a published state-of-the-art bidirectional algorithm does not prove that one does not exist, we implemented bidirectional heuristic search in three domains—the 15 Puzzle, the Pancake Problem, and Rubik's Cube—and compared it to unidirectional heuristic search. We compared BS*, as a canonical representative of front-to-end search, to A* on 1,000 random instances in each domain. The Pancake Problem instances have 50 pancakes and the Rubik's Cube instances were generated with 16 random moves, the hardest solvable within our memory limitations. In the 15 Puzzle, we observed that the standard additive PDBs are inconsistent, as they compress the location of the blank (Felner et al. 2007; Korf and Felner 2002); while this has been briefly noted (Breyer and Korf 2010), it appears to be generally overlooked. As we require a consistent heuristic, we made our 15 Puzzle PDBs consistent by not compressing away the blank tile. As a side-effect, this reduces node expansions at the cost of greater memory requirements.

These results are summarized in table 2. Shown are the average reduction in nodes expanded by A* over BS*, the average reduction in time of A* over BS*, and the percentage of instances where BS* beats A*.

Domain	A* Node Reduction	A* Time Reduction	BS* Wins
15 Puzzle	20%	37%	18%
Pancake Problem	11%	32%	16%
Rubik's Cube	15%	74%	29%

Table 2: A* vs BS* in three domains

In all three domains A* expands fewer nodes on average. BS* wins on some instances; importantly, though, BS* always expands at least as many nodes as A* with f cost less than C^*. It only wins by expanding fewer nodes with f cost *equal* to C^*. This was shown possible by Kaindl and Kainz, and is no different than the effect of changing the order of node exploration in a unidirectional search. Particularly when accounting for bookkeeping overhead, it is clear that a unidirectional search outperforms a bidirectional search when using a strong heuristic, as predicted.

4.2 Four-Peg Towers of Hanoi

One domain we examined is the four-peg Towers Of Hanoi with arbitrary start and goal states. In the standard game, all disks start and end on a single (different) peg, allowing only one problem instance for a given number of disks. With arbitrary start and goal states we can consider many more instances. In the standard instance an optimal solution can be found by searching to all possible midpoint states, where all but the largest disk are distributed among the middle two pegs; the second half of the solution is a mirror image of the first half with relabeled pegs (Hinz 1997). In a sense, this conducts half of a bidirectional search to the midpoint, with the other half of the search being obtained for free. With arbitrary start and goal states this symmetry is lost and we must do complete searches all the way to the goal.

We considered many random instances with 20 and 21 disks. We compared a bidirectional brute-force search to unidirectional heuristic search, using disk-based search for both, as described in (Korf 2008). Bidirectional search was performed using brute-force breadth-first search and unidirectional search was performed using BFHS with C^* as a cutoff. In general, the optimal solution cost is not known in advance, so this is an idealized algorithm. A more realistic solver would have to implement a more complicated algorithm—like disk-based A*—or use a more expensive, iterative-deepening approach to find optimal solutions.

For the heuristic we considered PDBs of various sizes. As we are using arbitrary goal states, a new PDB must be generated for each instance. While we found the savings of

[1]The state-of-the-art solver for Rubik's Cube and the Sliding Tile puzzle is a front-to-front bidirectional algorithm, SFBDS. As we address only front-to-end algorithms, however, this is outside the scope of our paper.

the heuristic function can outweigh the cost of generating the PDB, this introduces a degree of freedom: the size of the PDB to generate for a given instance.

For N=21 disks, we solved 10 random instances with both algorithms using three PDB sizes. The weakest PDB configuration partitions the disks into two sets of disks, one of size 15 and one of size 6; the strongest partitions them into sets of size 17 and 4. The results are given in table 3. The first row gives PDB creation time, which applies only to unidirectional searches and does not vary by instance. The remaining rows are the time required by each solver for each instance, including PDB creation. The best performance overall is bolded (always bidirectional search), and the best unidirectional search is underlined. Times are in seconds.

	Bidir. BF	Unidir. 15/6	Unidir. 16/5	Unidir. 17/4
PDB Creation	N/A	83s	328s	1,380s
Instance 1	**3,707s**	51,186s	20,770s	*8,315s*
2	**125s**	*202s*	411s	1,445s
3	**1,299s**	9,385s	3,338s	*1,858s*
4	**300s**	*461s*	474s	1,461s
5	**589s**	1,223s	*1,057s*	1,650s
6	**7,283s**	88,349s	50,967s	*24,261s*
7	**158s**	*272s*	466s	1,415s
8	**4,104s**	60,981s	26,803s	*7,660s*
9	**414s**	869s	*693s*	1,539s
10	**445s**	2,168s	*1,022s*	1,577s

Table 3: Instances of 21-disk Towers of Hanoi.

Bidirectional brute-force search outperformed the best unidirectional heuristic search by an average factor of 1.5-3. In some instances, like number three, unidirectional search is faster when ignoring PDB creation; when counted, however, bidirectional brute-force search is always fastest. If bidirectional brute-force search is compared to each unidirectional search independently, rather than against the strongest of the three for a given instance, the relative performance is often much stronger, sometimes by over an order of magnitude. As one would expect, the weaker heuristic does relatively worse on harder instances and the stronger heuristic does relatively worse on easier instances (due to PDB creation cost). We cannot know the difficulty of an instance in advance, and thus do not have the luxury of guaranteeing we will always use the best PDB.

In addition, we solved 100 random instances with 20 disks. We selected the heuristic by solving the first 10 instances with different PDBs and finding the configuration with the best overall performance, which was a partitioning into two PDBs of 15 and 5 disks. We then used this configuration to solve all 100 instances. Table 4 summarizes this data. The first two columns give the solution time for each algorithm, and the third gives the relative slowdown of unidirectional heuristic search over bidirectional brute-force search. The first row gives PDB creation time, the second gives the mean solution time over all 100 instances, and the third and fourth give the values for the instances with the worst and best relative performance of bidirectional brute-force search. All times are in seconds. Again, bidirectional brute-force search outperforms unidirectional heuristic search in all instances, by an average factor of 3.65.

	Bidir.	Unidir. 15/5	Slowdown
PDB Generation	N/A	83s	N/A
Mean Results	**217s**	793s	3.65
Best for BiDir.	**347s**	3939s	11.35
Worst for BiDir.	**85s**	126s	1.48

Table 4: 100 instances of 20-disk Towers of Hanoi.

The mean improvement of bidirectional brute-force over unidirectional heuristic search is roughly equal to the fraction of nodes expanded shallower than the midpoint, shown in table 1. This is as we expect if the contribution of a bidirectional brute-force search is to avoid expanding nodes that appear past the midpoint in a unidirectional search.

4.3 Work Spent Proving Optimality

Finally, we refute a statement from (Kaindl and Kainz 1997) that is often used to explain the ineffectiveness of bidirectional heuristic search: that front-to-end algorithms tend to find optimal solutions very early and spend a long time proving optimality (Edelkamp and Schrödl 2011; Felner et al. 2010; Lippi, Ernandes, and Felner 2012). This claim is separate from their proof of BHPA's ineffectiveness. In fact, in our bidirectional experiments, the optimal solution was found very late: on average after 90% of the nodes had been generated in the 15 Puzzle, after 86% had been generated in the Pancake Problem, and after 96% had been generated in Rubik's Cube. We used much stronger heuristics than Kaindl and Kainz, and so our results make sense: in a best-first search with a strong heuristic there are few nodes generated with high g cost and thus fewer opportunities for frontier intersections. This reinforces our expectation that pathological cases such as the one in figure 3 will rarely occur in practice.

5 Conclusions

We presented an intuitive explanation for the limited effectiveness of front-to-end bidirectional heuristic search. Bidirectional brute-force and unidirectional heuristic search both prevent expansion of nodes with high g cost. As such, for a given domain we would expect one of the techniques to dominate the other and for there to be no benefit from combining the two. We also show that there is no general proof of this claim, as there are pathological cases where a bidirectional heuristic search *is* the preferred algorithm.

We also considered a new variant on a well-studied domain: the four-peg Towers Of Hanoi with arbitrary start and goal states. We found that bidirectional brute-force search, surprisingly, is the state-of-the-art, outperforming unidirectional heuristic search. This data is predicted by our theory.

Finally, we found that an often-repeated claim explaining the poor performance of bidirectional heuristic search—that most of its time is spent proving solution optimality—is not true in general, in particular when using strong heuristics.

References

Barker, J. K., and Korf, R. E. 2012. Solving peg solitaire with bidirectional BFIDA*. In *Association for the Advancement of Artificial Intelligence*.

Breyer, T. M., and Korf, R. E. 2010. 1.6-bit pattern databases. In *Association for the Advancement of Artificial Intelligence*.

Culberson, J. C., and Schaeffer, J. 1996. Searching with pattern databases. In *Advances in Artifical Intelligence*. Springer. 402–416.

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

Edelkamp, S., and Schrödl, S. 2011. *Heuristic Search: Theory and Applications*. Morgan Kaufmann.

Felner, A.; Korf, R. E.; Meshulam, R.; and Holte, R. C. 2007. Compressed pattern databases. *Journal of Artificial Intelligence Reseach* 30:213–247.

Felner, A.; Moldenhauer, C.; Sturtevant, N. R.; and Schaeffer, J. 2010. Single-frontier bidirectional search. In *Association for the Advancement of Artificial Intelligence*.

Felner, A.; Zahavi, U.; Holte, R.; Schaeffer, J.; Sturtevant, N.; and Zhang, Z. 2011. Inconsistent heuristics in theory and practice. *Artificial Intelligence* 175(9):1570–1603.

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

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

Hinz, A. M. 1997. The tower of hanoi. *Algebras and Combinatorics: Proceedings of ICAC97* 227–289.

Kaindl, H., and Kainz, G. 1997. Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research* 7:283–317.

Korf, R. E., and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134(1):9–22.

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

Korf, R. E. 2008. Linear-time disk-based implicit graph search. *Journal of the ACM* 55(6):26.

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

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

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

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

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

Yang, F.; Culberson, J. C.; Holte, R.; Zahavi, U.; and Felner, A. 2008. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research* 32:631–662.

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