Predicting the Effectiveness of Bidirectional Heuristic Search

Nathan R. Sturtevant
Dept. of Computing Science
University of Alberta
Edmonton, AB, Canada
nathanst@ualberta.ca

Shahaf Shperberg
CS Department
Ben-Gurion University
Be'er-Sheva, Israel
shperbsh@post.bgu.ac.il

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

Jingwei Chen
Dept. of Computing Science
University of Alberta
Edmonton, AB, Canada
jingwei5@ualberta.ca

Abstract

The question of when bidirectional heuristic search outperforms unidirectional heuristic search has been revisited numerous times in the field of Artificial Intelligence. This paper re-addresses the question of when bidirectional search outperforms unidirectional search using an updated theoretical understanding of the problem. We show that a core set of critical states in the state space are the primary factor determining whether a bidirectional search can outperform a unidirectional search and provide simple measures to determine whether a state space and heuristic contains these critical states. We similarly discuss and show the impact that asymmetry in the underlying problem graph has on the performance of bidirectional algorithms. Experimental results show the impact of these factors on whether a problem should be solved using unidirectional or bidirectional search.

1 Introduction

In the past few years the understanding of bidirectional search has improved significantly, with new theory and new search algorithms. But, an outstanding question remains: when will bidirectional search be more effective than unidirectional search? There have been two recent answers to this question. Barker and Korf (2015) posited that with a strong heuristic, A* will always be preferred, but that with a weak heuristic, bidirectional brute force search will be preferred. Holte et al. (2017) refined this by looking at whether algorithms will expand states that are near, far, or remote from the start and the goal respectively.

There are four weaknesses in these analyses. First, both piece of work assume that the bidirectional search frontiers meet in the middle of the optimal path. While this is true for the MM algorithm (Holte et al. 2017), state-of-the-art algorithms like NBS (Chen et al. 2017) and DVCBS (Shperberg et al. 2019) will rarely meet in the middle, and thus the analyses do not fully apply to these algorithms. Second, Barker and Korf (2015) assume symmetric state spaces, which does not hold in practice. Third, these analyses predate the theory that fully describes the necessary expansions in bidirectional search (Eckerle et al. 2017b) that gives a fuller picture of bidirectional search. Finally, both analyses require solving problem instances to measure performance and do not provide inexpensive predictive measures to determine whether bidirectional search will outperform unidirectional search.

This paper addresses these weaknesses by providing an analysis which is directly based on the new theory of bidirectional search (Eckerle et al. 2017b; Chen et al. 2017; Shaham et al. 2017). In particular, the necessary expansions to solve a problem can be described through a bipartite *must-expand graph* (G_{MX}). The *minimum vertex cover* of G_{MX} (MVC) determines the minimum number of node expansions needed to solve a problem. If the MVC is bidirectional there exists a bidirectional algorithm that outperforms any unidirectional algorithm.

However, as with past analysis, building G_{MX} and finding the MVC is expensive. Therefore, we analyze the nature of the MVC to understand when it will be bidirectional. We then devise measures that predict whether the MVC will be unidirectional or bidirectional. These measures can be applied without solving a problem and without building G_{MX}. Experiments illustrate that these measures are predictive of the properties of the MVC and also that state-of-the-art bidirectional algorithms like NBS and DVCBS are able to outperform unidirectional A* when the MVC is bidirectional.

1.1 Definitions and Terminology

A shortest-path problem instance, I, is defined as a n-tuple $(G = \{V, E\}, start, goal, h_F, h_B)$. G is a graph, $start, goal \in V$ and the aim is to find the least-cost path (with cost C^*) between $start$ and $goal$. Unidirectional search algorithms search forward from the $start$ to the $goal$ and never expand the $goal$. Bidirectional search algorithms interleave two separate searches, a search forward from $start$ and a search backward from $goal$. f_F, g_F and h_F indicate f-, g-, and h-costs in the forward search and f_B, g_B and h_B similarly in the backward search. $d(x, y)$ denotes the shortest distance between x and y, so $d(start, goal) = C^*$. *Front-to-end* algorithms use two heuristic functions. The *forward heuristic*, h_F, is *forward admissible* iff $h_F(u) \leq d(u, goal)$ for all u in G and is *forward consistent* iff $h_F(u) \leq d(u, u') + h_F(u')$ for all u and u' in G. The *backward heuristic*, h_B, is defined analogously. Front-to-front

bidirectional search algorithms are outside the scope of this paper.

Let I_{AD} be the set of problems with admissible heuristics, and $I_{\text{CON}} \subseteq I_{\text{AD}}$ be the set of problems with consistent heuristics. An *algorithm* is admissible if it returns an optimal solution, but the admissibility of an algorithm must be qualified with the set of problems on which it is admissible. The analysis in this paper primarily concerns the performance of algorithms that are admissible on I_{AD} when solving problems in I_{CON}. We will discuss this further when introducing the theory of bidirectional search in Section 3.

2 Previous Bidirectional Analysis

Recent analysis of bidirectional and unidirectional search provided explanations of when each approach will work well. But, these analyses assume that the bidirectional search frontiers will meet in the middle of the optimal path and also require solving problem instances to measure properties of the given search tree. Thus, they are only useful *after* solving problem instances. We next provide an overview on these analyses.

2.1 Barker and Korf Conjecture

Barker and Korf (2015) (Denoted BK) defined a unidirectional heuristic as *weak* if it expands the majority of its nodes deeper than the solution midpoint ($\frac{1}{2}C^*$). A unidirectional heuristic is *strong* if the majority of states expanded are at shallower depth than the solution midpoint. Based on these two definitions they made the following conjectures: **BK1:** *Adding a weak heuristic to a bidirectional brute-force search cannot prevent it from expanding additional nodes.* **BK2:** *With a strong heuristic, a bidirectional heuristic search expands more nodes than a unidirectional heuristic search.*

These conjectures are under the assumption that the forward and backward searches are of roughly equivalent difficulty and with similar search trees, and that they meet in the middle. Furthermore, their definition of a weak and a strong heuristic depends on the problem instance being solved; no analysis is provided to determine if a heuristic is strong or weak without fully solving a given problem instance.

2.2 Holte General Rule 2

Holte et al. (2017) developed the MM bidirectional heuristic search algorithm which is guaranteed to *meet in the middle*. i.e., MM will never expand a state whose g-value exceeds $C^*/2$. This is achieved by MM's novel selection criterion and priority functions, $pr_F(u)$ and $pr_B(u)$ for the forward and backwards directions, respectively:

$$pr_F(u) = \max(g_F(u) + h_F(u), 2 \cdot g_F(u))$$
$$pr_B(v) = \max(g_B(v) + h_B(v), 2 \cdot g_B(v))$$

MM expands a state with minimum priority from either direction. They also described three general rules that characterize the performance of unidirectional and bidirectional heuristic search. One of them, denoted GR2, is most relevant to the analysis in this paper. GR2 relies on sets of nodes that are determined by their distances from the start and the goal. The set FF includes all states that A* might expand but MM will not. These are nodes with $g_F(n) > \frac{1}{2}C^*$ and $g_B(n) > \frac{1}{2}C^*$. The set RN includes all states MM will expand but A* will not. These are nodes with $g_F(n) > C^*$ and $g_B(n) \leq \frac{1}{2}C^*$. Then, GR2 is:

GR2: *When $|FF| > |RN|$, A* will expand more nodes than MM if the heuristic is weak but will expand fewer nodes than MM if heuristic is very accurate.*

This is in line with BK2; with a strong heuristic, A* tends to outperform bidirectional heuristic search. However, here too, it isn't clear what makes a heuristic very accurate or strong. Furthermore, a problem instance needs to be fully solved so as to know the size of the different sets in order to make a prediction based on these rules.

3 Bidirectional Theory

We base our predictive measures on the theory of bidirectional search, which is described in this section.

With an admissible heuristic, any admissible unidirectional algorithm meeting reasonable theoretical assumptions must expand all states with $f(n) < C^*$ in order to prove the optimal solution (Dechter and Pearl 1985). Eckerle et al. (2017b) generalized this to bidirectional search and showed that in bidirectional search necessary expansions are defined for pairs of states u and v in the forward and backward frontiers, respectively. The lb function measures the minimum cost path that can connect $start$ and $goal$ via u and v.

Definition 1. *For each pair of states (u, v) let*

$$lb(u, v) = \max\{f_F(u), f_B(v), g_F(u) + g_B(v)\}$$

In bidirectional search, a pair of states (u, v) is called a *must-expand pair* if $lb(u, v) < C^*$. However, in a must-expand pair only one of u or v must be expanded, not both. An algorithm that neither expands u nor expands v when $lb(u, v) < C^*$ may miss the optimal solution. The analysis in this paper is purely with respect to necessary expansions.

3.1 The Must-Expand Graph (G_{MX})

The unique property of the must-expand condition is that it contains an *or* between the nodes, also a feature of the vertex cover problem. Chen et al. (2017) showed how to represent the necessary expansions conditions as a vertex cover on a *must expand graph* denoted by G_{MX}:

Definition 2. *The Must-Expand Graph G_{MX} on a problem instance is an undirected, unweighed bipartite graph. For each state $s \in G$, there are two vertices in G_{MX}, the left vertex s_F representing the state in the forward direction and the right vertex s_B representing the state in the backwards direction. For each pair of states $m, n \in G$, there is an edge in G_{MX} between m_F and n_B if and only if (m_F, n_B) is a* must-expand pair.

It follows that the minimum number of node expansions required to solve a problem instance by bidirectional heuristic search is the size of the *minimum vertex cover* of G_{MX}, denoted hereafter as MVC.

Definition 1 is valid for problems in I_{CON} with algorithms that are admissible on I_{AD}. Alternate definitions of lb (or

equivalent functions) exist for state spaces with $\epsilon > 0$ edge costs (Shaham et al. 2018), for state spaces with consistent heuristics (Shaham et al. 2018), and for front-to-front search (Eckerle et al. 2017b).

The analysis here is presented only for the simplest case. It can be trivially extended to ϵ edge costs. The extension to algorithms that are admissible on I_{CON} but not on I_{AD} (Alcázar, Riddle, and Barley 2020) is an important matter of future work, as this information can be used to significantly improve performance. But, the high-level point applies:[1] The MVC of the appropriate G_{MX} graph describes the minimal node expansions required to prove an optimal solution to any problem.

Figure 1(a) shows a G_{MX} graph for a sample 20-pancake problem instance with $C^* = 13$ with the GAP heuristic (Helmert 2010). Each vertex is labeled internally with its g-cost. The left vertices are sorted by increasing g_F-costs, while right vertices are sorted by decreasing g_B-cost. Additionally, nodes with the same g_F or g_B are merged into a single (weighted) vertex, and the *weight* of that vertex, the total number of merged nodes, is written adjacent to the vertex. In this ordering, each horizontal pair of vertices u and v have $g_F(u) + g_B(v) = C^* - 1$, which, in Figure 1(a), is 12. For simplicity the figure draws a full edge from a left node u to a right node v only if u has no edges to right nodes with g_B larger than $g_B(v)$; similarly for right nodes. Only the prefix of other edges is shown. For example there is a full edge in the figure from the node with $g_F = 5$ to the node with $g_B = 7$ but this node is also connected to all nodes with $g_B < 7$ as shown by the prefixes of edges.

3.2 MVC of GMX

Let t_F and t_B be thresholds such that $t_F + t_B = C^* - 2$. There is a family of contiguous vertex covers (VCs) for all such pairs (t_F, t_B), where all nodes with $g_F \leq t_F$ in G_{MX} are included in the forward direction of this VC, and all nodes with $g_B \leq t_B$ are included in the backward direction of this VC. It was proven that one such (t_F, t_B) partition is the MVC (Shaham et al. 2017) and is denoted by (t_F^*, t_B^*). The number of nodes of the (t_F, t_B)-VC partitions is determined by summing up the weights of the nodes with $g_F \leq t_F$ and with $g_B \leq t_B$. These costs are shown in Figure 1(a)–1(c) at the *crossing point*, i.e., right between the respective values of t_F and t_B. For example, consider Figure 1(b). In the figures, the nodes of the MVC partition are colored. For this example the MVC partition is $((t_F^*, t_B^*) = (4, 7))$ and include 776,458 nodes. We note that a forward unidirectional partition, (forward VC), is $(12, \varnothing)$ (1,672,402 nodes) with a crossing point above the graph and a backward unidirectional partition, (backward VC), is $(\varnothing, 12)$ (819,651 nodes) with a crossing point below the graph.

For simplicity we will typically assume, without loss of generality, that the forward unidirectional partition (i.e., a $(C^*-1, \varnothing)$ partition) includes fewer nodes than a backwards unidirectional partition (a $(\varnothing, C^* - 1)$ partition). Thus, the start state is always included in MVC. The aim of this paper is to provide insight on when (t_F^*, t_B^*) is *non-null* (i.e., $t_F^*, t_B^* \neq \varnothing$).

3.3 Fractional MM (fMM(p))

fMM(p) (Shaham et al. 2017) is a generalization of MM that never expands nodes in the forward direction whose g-value exceeds C^*/p, and never expands nodes in the backward direction whose g-value exceeds $C^*/(1-p)$. For a given fraction $0 < p < 1$, fMM(p) chooses a node for expansion according to the following priority functions:

$$pr_F(u) = \max(g_F(u) + h_F(u), g_F(u)/p)$$
$$pr_B(u) = \max(g_B(u) + h_B(u), g_B(v)/(1-p))$$

Shaham et al. (2017) showed that for every problem instance, there exists a fraction $p^* = t_F^*/(t_F^* + t_B^* + 1)$ such that fMM(p^*) is *optimally efficient* and will expand exactly the MVC of G_{MX}. Thus, in theory, not only we can know whether a problem instance is bidirectional or not but we also have an algorithm that will expand exactly the minimal number of nodes required to guarantee the optimality of its solution. In practice, however, this can only be done after C^* is known and G_{MX} is built. Furthermore, there are no guarantees if fMM does not search with p^*. Thus, in the rest of this paper we look for a method to predict whether the MVC is bidirectional or not without building G_{MX}.

4 Conditions for a Bidirectional MVC

Given the bidirectional theory, the following definitions can be used to classify whether bidirectional search is the best approach for a problem instance.

Definition 3. *A problem instance is* bidirectional *if at least one MVC of G_{MX} contains vertices on both sides of G_{MX}.*

Definition 4. *A problem instance is* unidirectional *if at least one MVC of G_{MX} only contains nodes on one side of G_{MX}.*

Definition 5. *A problem instance is* weakly bidirectional *if it is both bidirectional and unidirectional.*

The problem instance in Figure 1(c) is weakly bidirectional because there are two possible MVCs which result in the same number of node expansions, one is bidirectional $(11, 0)$ and one is unidirectional $(12, \varnothing)$.

If we know the full structure of G_{MX} we can determine whether the problem instance is unidirectional or bidirectional. However, since G_{MX} is not known in advance and is costly to build, we aim instead to predict beforehand whether the problem instance is bidirectional or unidirectional. To do this, we identify a set of *critical states* (defined below) that impact this prediction. We then show how we can test for these critical states.

We begin with an analysis of the three G_{MX} graphs of the 20-pancake problem in Figure 1 each having a different heuristic. Consider Figure 1(a) with the GAP heuristic. The *critical states* for this problem have g_F-cost between 8-12 and are marked with a red box. The key feature here is that the weight of these vertices is 0. In a unit-cost domain like the pancake puzzle, there must be states in the solution with all g-costs between 0 and C^*. But, if these states have perfect

[1] The statement only applies to deterministic and expansion-based algorithms (denoted as DXBB by Eckerle et al. (2017a)), which is a standard assumption for heuristic search algorithms.

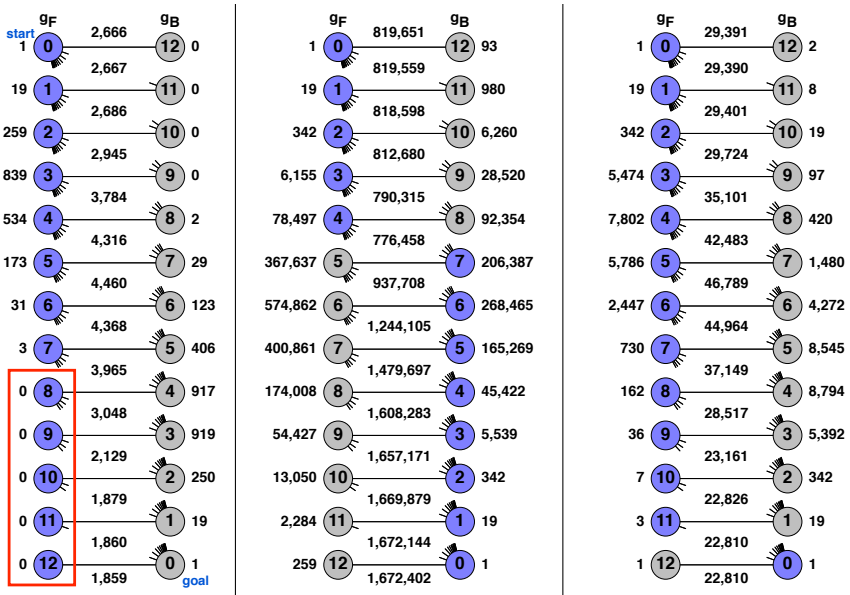


Figure 1: G_{MX} figures for a 20-Pancake instance

heuristics, they will not be part of G_{MX}, since states must have $f < C^*$ to appear in G_{MX}.

Recall that an MVC has two thresholds, t_F^* and t_B^*. Given the unidirectional $(12, \varnothing)$-VC in Figure 1(a) (1859 nodes), the question is whether using a larger t_B could improve the VC. The $(11, 0)$-VC, includes one additional node in the backwards direction (the goal) but there is no reduction of nodes in the forward direction. The $(10, 1)$-VC adds 19 nodes in the backwards direction, again with no savings in the forward direction. The first VC that reduces the nodes in the forward direction is the $(6, 5)$-VC, which has 2,512 backward nodes (sum of levels 0–5) in order to reduce 3 forward nodes (sum of levels 7–12). So, the unique MVC is unidirectional in Figure 1(a) because the states in G_{MX} with g-cost close to C^* all have perfect h_F-values or high enough h_F-values that exclude them from G_{MX}. The large number of nodes in G_{MX} with weight of 0 make it difficult for a bidirectional search to outperform a unidirectional search.

Now consider Figure 1(b) (GAP\2 heuristic where gaps involving tokens 1 and 2 are omitted from the heuristic). In this example, all vertices in G_{MX} have weights > 0. Including the first backward node in the $(11, 0)$-VC already saves 259 forward nodes (a net gain of 258). This implies that all MVCs are bidirectional when there are many states in G_{MX} with g-cost close to C^* with inaccurate heuristics.

Note that to be included in G_{MX}, the states with $g_F = 12$ must have $h_F < 1$, the states with $g_F = 11$ must have $h_F < 2$ and similarly for other states. Thus, the two properties of the critical states are that they (1) have large g_F-cost and (2) have inaccurate and small h_F-cost so that their f-value is $< C^*$ and they appear in G_{MX}. Trivially, states having perfect h-cost on the optimal path are not critical, and they will not appear in G_{MX}.

Definition 6. *Critical states for $k < C^*$ are those that have $g_F > C^* - k$ and $h_F < C^* - g_F$ (or symmetrically in the backwards direction).*

This leads us to our main claim:

Corollary 1. *If there exists some k such that there are more critical states for k than states with $g_B < k$ in the backward direction of G_{MX} (those with $h_B < C^* - g_B$) then all MVCs will be bidirectional.*

Note that this definition is similar to the BK definition of a weak heuristic for $k = C^*/2$. But, under the BK formulation, the heuristic in Figure 1(b) is strong – the majority of states expanded are found in the first half of each of the unidirectional searches – and so unidirectional search should outperform bidirectional search by their reasoning. (C^* is 13, so the g-costs 0-6 are in the first half of the search.) In this example there are enough critical states even for $k = 1$ (259 states) such that the MVC is bidirectional.

4.1 Measuring Critical States

The definition of critical states depends on both the g-cost and h-cost of states. While h-costs can be sampled in general, g-costs are instance dependent. However, states near the goal must have g_F-costs that are close to C^*. Thus, we propose two measurements that estimate the existence and frequency of critical states in the state space. The first measurement (M1) samples states near the goal to see what percentage of them have inaccurate heuristics. The second measurement (M2) is the ratio between the total number of states with low heuristic values that are not near the goal and the total states near the goal. The larger the ratios returned by M1 and M2 the more potential critical states there are. But, because we cannot measure g_F, we cannot guarantee that

they are critical states in practice. The more states that have the potential to be critical states, the greater the likelihood that some of them are and the MVC is bidirectional.

The procedure for M1 is relatively simple and is computationally inexpensive. M1 does a backwards Dijkstra search from the goal up to a distance (radius) r_{M1}, counting the total number of states seen with $g_B \leq r_{M1}$. The number of such states is denoted by $s_{g \leq r_{M1}}$. M1 also counts the total number of states seen with both $g_B \leq r_{M1}$ and $h_F \leq r_{M1}$; the number of these states is denoted by $s_{h \leq r_{M1}}$. M1 is then reported as the percentage of states seen with inaccurate heuristics, i.e, $M1 = s_{h \leq r_{M1}} / s_{g \leq r_{M1}}$. If M1 is 100% then the MVC will be at least weakly bidirectional because for $r_{M1} = 0$ there is at least one critical state — the last node on the optimal path before the goal.

There are several procedures that can be used to compute M2. The most general procedure is to sample random states in the state space and measure whether the heuristic is less than some fixed value h_{M2}. From this sample we can estimate the total number of states with $h_F(s) \leq h_{M2}$ in the entire state space, $\hat{s}_{h \leq h_{M2}}$. If the search space size $|S|$ of the given problem is known, as in the domains examined in this paper, $\hat{s}_{h \leq h_{M2}}$ can be estimated by sampling k random states, and counting those with $h_F(s) \leq h_{M2}$ (denoted by $k_{h \leq h_{M2}}$); formally, $\hat{s}_{h \leq h_{M2}} = |S| \cdot \frac{k_{h \leq h_{M2}}}{k}$. Otherwise, if search space size is unknown, an estimation of $|S|$ (Lelis, Stern, and Sturtevant 2014) can be used. In addition, $s_{h \leq h_{M2}}$ and $s_{g \leq h_{M2}}$ are defined analogously to $s_{h \leq r_{M1}}$ and $s_{g \leq r_{M1}}$. M2 is then the ratio between the states with inaccurate heuristics farther from the goal to the number of states near the goal, i.e., $M2 = (\hat{s}_{h \leq h_{M2}} - s_{h \leq h_{M2}}) / s_{g \leq h_{M2}}$.

The accuracy of sampling depends on the number of samples and the size of the search space. But, for many types of heuristics, M2 can be measured more precisely. Consider, for instance, a grid world with straight-line distance as a heuristic. For a given goal state we can directly count $S_{h \leq h_{M2}}$ and $s_{h \leq h_{M2}}$ by iterating over the local neighborhood of the goal. This is called *direct sampling*.

In pattern databases (PDBs) (Culberson and Schaeffer 1996), the PDB stores the distance from each pattern to the goal, where a pattern abstracts away some state literals. In Rubik's Cube, a PDB might abstract all edge cubes and only store the distances for configurations of corner cubes. Although there is only one entry in the PDB with the value 0 in this heuristic, there are $12! \times 2^{11}$ possible edge cube configurations for each configuration of corner cubes. Thus, once M1 is measured near the goal, M2 can be computed from the PDB distribution in the original state space. The procedure is more complex when the max or sum of multiple PDBs is used (Korf and Felner 2002; Holte et al. 2006), but it is still possible to perform similar direct measurements of $S_{h \leq h_{M2}}$ in this situation. For instance, assume we are taking the sum of PDBs p_1 and p_2, where the patterns for p_1 and p_2 are disjoint and cover all variables in the PDB. Then, we can find the states in each PDB with $h \leq h_{M2}$. Finally, we can test if the disjoint patterns can be combined into a legal state s with $h_{p_1}(s) + h_{p_2}(s) \leq h_{M2}$.

4.2 Asymmetry

As mentioned before, the BK analysis assumes that the difficulty of solving a problem in the forward and backwards directions is approximately the same. They do note, however, that if one direction is far more difficult than the other, bidirectional search may still perform well.

Expanding on this, consider a road network, where the density of roads in the city is much larger than the density in the countryside. If the start state is in the middle of a city and the goal in the countryside, it may be more efficient to search in reverse towards the start state, as the reverse search can terminate when it reaches the start, avoiding extra search inside the city. Although the best unidirectional search may always outperform the best bidirectional search, if the best search direction isn't known, a bidirectional search may still, on average, outperform an unidirectional search with an arbitrary direction.

When the MVC is unidirectional, bidirectional algorithms like NBS (Chen et al. 2017) will do at most twice the work of the best unidirectional search. Because the best direction for A* search is not known, on expectation A* (with an arbitrary direction) will do the average of the two possible unidirectional searches. Thus, on symmetric problems with similar-sized forward and backwards searches and low M1 and M2 measurements, A* will always expand close to the MVC as in Figure 1(a).

With asymmetric forward and backwards searches the average of the nodes in the two directions will be larger than the best unidirectional VC. Thus, the difference between twice the minimum unidirectional search and the average forward and backwards search will be reduced, improving the performance of bidirectional search relative to unidirectional search even without a bidirectional MVC. If all MVCs are bidirectional, bidirectional algorithms are expected to have an even larger advantage over unidirectional algorithms. Thus, we propose a third measure of asymmetry, M3. M3 samples states in the state space and performs U Dijkstra expansions from each sampled state, recording the maximum g-cost expanded. Intuitively, if the maximum g-cost seen differs in each search, then the state space is more asymmetric. Let ℓ be the number of samples. Let x be the most common maximum g-cost that was seen in these samples. Let m be the number of times x was seen. Then M3 is $1 - m/\ell$. So, if, in 10 samples the same maximum g-cost is seen, M3 will be 0%. If a different g-cost is seen in each sample, then M3 will be 90%.

4.3 Summary

We summarize the impact of these measures with rules:

Rule 1: If M1 is high (close to 100%) and M2 and M3 are low, there are only critical states near the goal. Thus, the problem instance will be weakly bidirectional, or close to that, i.e., the difference between the unidirectional VC (UVC) and the MVC will be small.

Rule 2: If M2 is large there are many critical states and all MVCs are expected to be bidirectional.

While rules 1-2 predict whether MVCs will be bidirectional, rule 3 predicts the performance of bidirectional search algorithms.

Rule 3: If M3 is high (greater than 50%), the number of expansions by bidirectional algorithms such as NBS will be much less than 2x the expansions of an arbitrary A* even if M1 and M2 are low and the MVC is unidirectional.

With regard to sampling, we recommend choosing some $h_{M2} < C^*/2$ which is as large as can be sampled efficiently on a given domain/heuristic combination.

4.4 Limitations

The measures and rules proposed in this paper are the first measures that do not require solving a problem to estimate whether the MVC is bidirectional. However, they are still estimates, and subject to the following weaknesses.

First, for any r_{M1} or h_{M2} used for sampling, an adversary could structure a problem such that all critical states are found for a parameter that is slightly larger. This is unavoidable without stronger assumptions about the problem structure, which are beyond the scope of this work.

Second, it is not possible to precisely place a threshold on M2 for when all MVCs will become bidirectional, as this also depends on other factors such as the branching factor and the solution depth. This can be partly mitigated by comparing M2 over different heuristics, as larger values are better. While our experimental results suggest that M2 > 9 is sufficiently high, this will not be universal for every possible domain. After our primary experimental results we will return to the impact of solution depth in more detail.

Despite these limitations, as we will show next, we can still make good predictions using M1-M3 and rules 1-3.

5 Bidirectional Predictions

To validate our analysis, we experiment with different heuristics in four domains: the 12 pancake puzzle, 4-peg towers of Hanoi (TOH4), grid maps and road maps. Columns C–E reported in Table 1 empirically evaluate M1-M3 on the four domains. In this table, we report average measurements over 50 random TOH4 instances, 50 hard 12-pancake instances (Valenzano and Yang 2017), 100 random road map instances on the map of Colorado, 21.7K instances of DAO grid maps, and 32.3K instance of grid mazes with maze corridor width $\in \{1, 4, 16\}$. All grid maps are taken from the MovingAI benchmarks (Sturtevant 2012).

In the pancake puzzle we use the GAP heuristic (Helmert 2010), which counts the gaps between adjacent tokens in the puzzle, and three variants of the GAP heuristic. GAP$\backslash X$ (Holte et al. 2017), ignores any gaps that involve the X smallest pancakes. GAP$-X$ subtracts X from the GAP heuristic (returning $\max(\text{GAP} - X, 0)$. GAP$\times X$ returns the GAP heuristic times X. In TOH4 we use two additive pattern databases of different sizes. The $M - N$ heuristic adds a heuristic for the larger M disks to a heuristic for the smaller N disks. In grid problems we use the octile heuristic, which is the shortest distance between two points on an 8-connected grid. In the road network instances we use Euclidean distance if the edge costs are measured by distance, and Euclidean distance divided by the maximum speed if edge costs are measured by travel time. We also use no heuristic in several domains to compare between bidirectional brute force and bidirectional heuristic search.

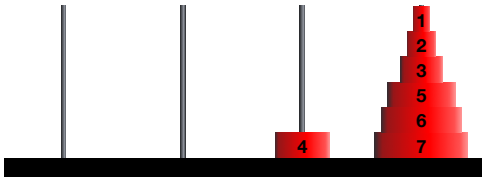


Figure 2: A TOH4 state for which M2 is large.

The goal of the experiments is to use a broad range of heuristics to illustrate the relationship between the MVC and M1–M3 along Rules 1–3, not to establish the state-of-art performance. For this reason we do not report runtimes in our results. The state-of-art performance and runtimes will change over time with new heuristics and new hardware, but the measures proposed here will not.

For each domain and heuristic combination, we measured M1 and M2 using $r_{M1} = 5, h_{M2} = 5$, where the number of nodes with inaccurate low heuristic ($S_{h \leq h_{M2}}$) for the pancake puzzle and the road network was collected by sampling. In TOH4 and grid maps we used the *direct sampling* procedure. M3 was measured using 1,000 samples, where each sample expanded 100 nodes. r_{M1} and h_{M2} are sufficient for the problems tested here. In state spaces such as the sliding-tile puzzle, where the branching factor is relatively low and the solution length is longer, larger values would be more appropriate, as the first few levels of the search tree contain only a few nodes. Note that as stated in Section 4.4, optimal values of r_{M1} and h_{M2} cannot be known a priori.

Columns F–J show quantities of the G_{MX} structure: percentage of instances in which the MVC is strictly bidirectional (labeled BMVC, column F), i.e., smaller than two unidirectional forward VC and backward VC, the number of nodes in the forward and backward VC's (FVC and BVC, columns G,H), the *minimum per instance* among the forward and backward VC's (I), and the size of the MVC (J). Columns K–N report the total number of node expansions (the MVC is only measuring necessary expansions) by state-of-the-art algorithms, which will be discussed in the next section. We bold the MVC size if it is smaller than the minimum UVC, meaning that the bidirectional MVC is smaller than the best UVC. We also bold the search algorithm with the fewest node expansions.

We begin by analyzing rule 1 and 2 on columns A–J.

Rule 1 Rule 1 states that when M1 is high and M2 and M3 are not, then the problem instance will be weakly bidirectional. If all MVCs are bidirectional they will not be significantly smaller than the best UVC. This result is best illustrated in the pancake puzzles with GAP$\times 0.9$. With this heuristic 56% of the problems solved have a bidirectional MVC, but the average UVC (224) is only 4% larger than the average MVC (216). With the GAP$\times 0.8$ heuristic the percentage of problems with a bidirectional MVC decreases, but M2 also increases, so M2 is no longer low. As a result the average UVC is 8% larger than the best bidirectional MVC,

A	B	C	D	E	F	G	H	I	J	K	L	M	N
Domain	Heuristic	M1	M2	M3	BMVC	FVC	BVC	Min UVC	MVC	A*	Rev-A*	NBS_ϵ	$DVCBS_\epsilon$
12-Pancake	GAP	26.4%	4.9	0.0%	0%	10	10	**8**	8	30	**28**	101	54
	GAP\1	95.9%	82.0	0.0%	64%	1,102	1,233	954	**832**	1,124	1,267	864	**841**
	GAP\2	97.6%	656.9	0.0%	96%	30,794	39,135	28,371	**12,466**	32,134	40,876	9,528	**8,011**
	GAP\3	98.7%	2,446.8	0.0%	100%	367,374	454,242	322,294	**48,994**	379,684	477,463	32,802	**25,080**
	GAP−1	100.0%	34.2	0.0%	74%	228	236	205	**187**	229	237	241	**174**
	GAP−2	100.0%	160.0	0.0%	100%	3,062	3,159	2,895	**1,643**	3,085	3,183	1,353	**1,269**
	GAP−3	100.0%	596.7	0.0%	100%	28,853	29,366	27,844	**8,406**	29,209	29,717	6,441	**5,290**
	GAP×0.9	100.0%	5.1	0.0%	56%	248	255	225	**216**	249	256	229	**204**
	GAP×0.8	100.0%	34.2	0.0%	48%	2,149	2,212	2,056	**1,904**	2,150	2,213	1,755	**1,712**
	GAP×0.7	100.0%	160.0	0.0%	88%	16,736	16,985	16,347	**11,744**	16,737	16,986	9,708	**9,330**
TOH4(12)	10+2	49.3%	3.8	0.0%	0%	64,264	68,149	**59,153**	59,153	**64,334**	68,173	100,080	69,010
	8+4	21.1%	9.0	0.0%	34%	456,156	456,439	420,749	**382,390**	457,401	459,373	**411,085**	434,347
	6+6	3.6%	10.9	0.0%	86%	772,889	796,639	697,965	**463,586**	789,603	806,767	**446,603**	525,811
	4+8	0.8%	9.6	0.0%	66%	530,936	547,752	480,027	**406,480**	548,850	568,615	**411,212**	427,702
	2+10	0.0%	3.9	0.0%	2%	162,656	174,659	152,421	**152,384**	**172,088**	190,364	199,880	192,271
	Zero	99.5%	78,141.6	0.0%	100%	8,262,691	8,560,419	7,826,880	**476,455**	8,262,691	8,560,419	450,539	**425,578**
Grids DAO	Octile	1.8%	0.0	66.7%	1%	9,525	9,222	7,785	**7,754**	9,646	**9,339**	12,136	9,815
	Zero	98.3%	504.7	66.7%	100%	19,448	19,935	17,760	**15,167**	19,466	19,955	16,819	**15,946**
Grids Mazes[1]	Octile	49.8%	1.9	94.7%	90%	63,657	63,694	49,257	**31,193**	63,660	63,698	**36,029**	49,298
	Zero	92.3%	10,081.4	94.7%	100%	71,455	71,571	57,961	**31,263**	71,474	71,590	36,034	**31,840**
Grids Mazes[4]	Octile	3.8%	0.1	95.7%	69%	97,215	97,072	77,765	**62,449**	97,228	97,084	**70,186**	90,301
	Zero	97.5%	5,230.7	95.7%	100%	114,084	114,111	94,678	**63,540**	114,105	114,133	70,259	**65,306**
Grids Mazes[16]	Octile	0.0%	0.3	88.8%	29%	125,496	126,236	108,282	**103,841**	125,538	126,278	128,572	**124,038**
	Zero	98.4%	3,843.4	88.8%	100%	144,773	145,445	126,898	**109,795**	144,801	145,473	129,310	**118,757**
Road Maps Distance	ED	99.5%	1.44	99.8%	16%	72,119	69,461	47,702	**47,666**	72,119	69,461	**69,110**	69,461
	Zero	99.2%	3,118.3	99.8%	100%	226,041	229,400	184,419	**129,484**	226,041	229,400	**143,428**	160,940
Road Maps Time	ED / speed	98.5%	10.5	99.8%	95%	133,147	128,165	93,645	**80,456**	133,148	128,166	**97,889**	128,165
	Zero	98.5%	6,749.3	99.8%	100%	230,075	227,785	181,105	**104,952**	230,076	227,786	119,873	**111,329**

Table 1: Evaluating the critical state measurements and algorithm performance on a variety of domains. Measures G-J are theoretical measures reporting necessary node expansions, while algorithmic results (K-N) report all node expansions.

a larger difference than with GAP×0.9.

Rule 2 Rule 2 states that when M2 is high, the MVC will be bidirectional. This rule is true for all brute-force searches (zero heuristic), and in other domain/heuristic combinations when M2 is greater than 9. More importantly, when comparing different M2 across two different heuristics, the heuristic with larger M2 has a smaller MVC relative to the best UVC, although there are some cases the difference between the bidirectional MVC and the UVC are relatively small. For instance, with the GAP×0.8 heuristic, in which M2 is 34.2, the bidirectional MVC is only 8% smaller than the UVC. Similarly, in the DAO maps with a zero heuristic the bidirectional MVC is only 15% smaller than the UVC.

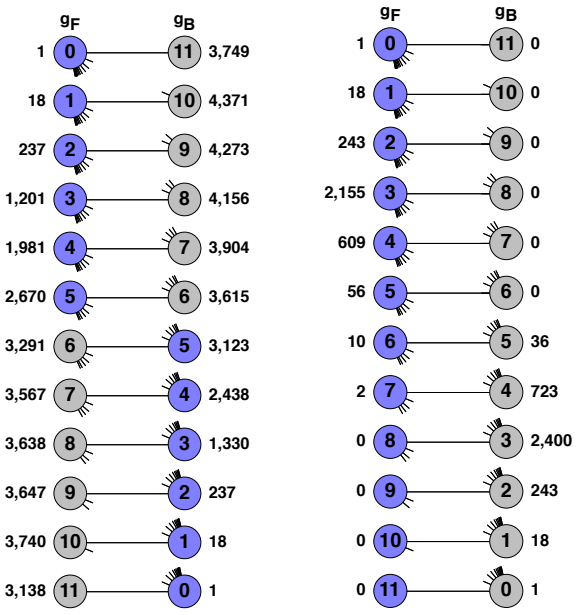
In TOH4 the 6+6 PDB is the only heuristic/domain combination with large M2 and small M1. It may not be obvious where the errors measured by M2 arise, so we illustrate them with an example in Figure 2. In this example, assume that a 4+3 PDB is being used, and that the goal is to stack all disks on the last peg. The 4-disk PDB will have a value of one, because in the 4-disk state space (ignoring the top 3 disks) it only takes one move to reach the goal. The 3-disk PDB will have a heuristic of zero because all disks are in the goal position. Thus, the heuristic in this state is very small, yet it will take many moves to reach the goal, which is measured by M2. The 6+6 PDB has many such states with small heuristic values, so M2 is high. Since very few of them are near the goal, M1 is low. Thus, we see that M2 is sufficient on its own to correctly predict a MVC which is bidirectional.

No Rule It is also useful to analyze domain/heuristic combinations where none of our rules apply. For instance, in TOH4 with the 2+10 heuristic M1-M3 are all small. Thus, as expected, only 2% of the problem instances are bidirectional. Similar analysis holds for the GAP heuristic.

6 Algorithmic Comparisons

Next, we move to solving problems with bidirectional and unidirectional algorithms. The goal is to evaluate whether a bidirectional MVC corresponds to better performance for a bidirectional search algorithm and to evaluate Rule 3. For a bidirectional search algorithm we experiment with Near-Optimal Bidirectional Search (NBS) (Chen et al. 2017) and Dynamic Vertex Cover Bidirectional Search (DVCBS) (Shperberg et al. 2019). The necessary expansions by NBS are guaranteed to be no more than twice the size of the MVC. DVCBS dynamically estimates the structure of G_{MX} and attempts to build a vertex cover at runtime, but has no guarantees on worst-case performance. The number of nodes expanded when finding a solution are found in Table 1 (Columns K – N) for forward A*, Reverse A*, NBS and DVCBS. In state spaces with unit edge costs we use the variants of these algorithms that perform better given the known smallest edge cost $\epsilon = 1$.

In these results the performance of bidirectional search algorithms like NBS and DVCBS are best when the MVC is bidirectional. Thus, a bidirectional MVC is predictive of good performance by bidirectional algorithms. Note that in these experiments NBS and DVCBS can even perform fewer



(a) One stronger heuristic (b) Max of weaker heuristics

Figure 3: G_{MX} for Rubik's cube: one vs several heuristics.

expansions than those needed for the MVC because of their use of ϵ (Shaham et al. 2018).

Rule 3 Rule 3 predicts that if M3 is high, then due to the asymmetry, bidirectional search algorithms like NBS will have better performance even if M1 and M2 are low. In the Pancake puzzle with the GAP heuristic, NBS is much worse than A* (3x total expansions and 2x necessary expansions). But, in the DAO maps with the octile heuristic, although only 1% of the problems have a bidirectional MVC, NBS is only 30% worse than A*. This is attributed to Rule 3 and the asymmetry in the domain. In maze grids with corridor size 16, M1 and M2 are both low, but NBS still has comparable performance to A* due to the problem asymmetry.

6.1 Bidirectional Heuristic Search

Barker and Korf (2015) conjectured that bidirectional heuristic search would never outperform unidirectional heuristic search or bidirectional brute-force search. There are several examples where this conjecture is violated in our experiments - where NBS performs fewer node expansions than A* and bidirectional brute force search. This occurs in TOH4 with the 8+4 and 4+8 heuristics and in road maps with time-based edges (where M1–M3 are all large). While these heuristics do not currently achieve state-of-art performance in these domains, they are still counter-examples to the conjecture.

7 Comparing Heuristics Strength

In the previous section we experimented across many domains and heuristics, ignoring the memory required by the different heuristics. In this section we consider the choice of one large heuristic versus many small heuristics.

Holte et al. (2006) showed that it is better for unidirectional search to take the maximum of a number of weak heuristics than to use one stronger heuristic. They explained this by the fact that the maximum of many heuristics tends to have fewer low h-values then the single stronger heuristic, and that these improved low h-values were the most important in the search.

In bidirectional search this is exactly the opposite. Low h-values correspond to critical states which lead to a bidirectional MVC. Here, using the single strong heuristic can be better because bidirectional search can avoid expanding states with low h-values and benefit from the high h-values.

We demonstrate this in Table 2 on the pancake puzzle and Rubik's cube. While GAP is a memory-free heuristic, we can simulate the use of different sizes PDBs by comparing a single GAP\2 heuristic to the maximum of four GAP\3 heuristics (GAP\3-MAX(4)). In Rubik's cube we compare one 7-edge PDB to the maximum of six 6-edges PDBs (6edges-MAX(6)). The later replicates the experiment performed by Holte et al. (2006), while adding results on bidirectional search and the VC of G_{MX}. Table 2 shows experiments on the same 50 pancake instances as used previously and 50 Rubik's Cube problem instances built by a random walk length 12. The average of the following quantities are also reported: BMVC, the forward and backward VCs, Min UVC, the MVC, and the ratio between the min UVC and the MVC.

For the 12-pancake problem, observe that the strength of both heuristics are very similar in terms of the necessary expansions (MVC). For GAP\2 96% of the problems have a bidirectional MVC. The average Min UVC is more than twice the size of the average MVC. However, for GAP\3-MAX(4) only 8% of the problems have a bidirectional MVC and the average Min UVC is almost identical to the size of the average MVC.

In Rubik's cube, the maximum of six 6-edge PDBs dominated the single 7-edge PDB for both unidirectional and bidirectional search. But, while unidirectional search is close to optimal with the maximum of weaker heuristics, it requires more than twice the MVC node expansions with the single larger heuristic which has many critical states.

Figure 3 shows sample G_{MX} graphs for a Rubik's cube instance. With the single larger heuristic there are many critical states and the MVC is bidirectional, while with the max of many small heuristics the MVC is unidirectional and there are no critical states in G_{MX}.

Thus, the choice of heuristics can have an important impact on whether bidirectional search performs well on a problem instance. The heuristics that work well for bidirectional search may have many critical states, as these can be avoided in a bidirectional search, while maximizing over several different heuristics helps avoid critical states, helping unidirectional search. A larger study is needed to examine which approach is more memory efficient. In particular, it is unclear whether a set of very small heuristics directly targeting critical states would have significant impact.

Domain	Heuristic	BMVC	FVC	BVC	Min Uni VC	MVC	$\frac{\text{Uni VC}}{\text{MVC}}$
12-Pancake	GAP\2	96%	30,794	39,135	28,371	12,466	2.28
	GAP\3-MAX(4)	8%	13,715	13,354	12,465	12,188	1.02
Rubik's Cube	7edges	100%	60,912	65,201	46,175	20,085	2.30
	6edges-MAX(6)	62%	4,090	4,012	3,903	3,885	1.00

Table 2: Comparing maximum of several weak heuristics over one stronger heuristic for uni- and bi-directional search.

Cost	A*	Rev-A*	Best-UNI	DVCBS	NBS	ratio
40-49	625,329	392,144	378,185	521,638	517,062	1.53
50-54	3,517,218	2,870,299	2,549,486	3,273,345	3,365,651	1.50
55-59	17,179,068	10,455,729	9,562,378	11,92,4392	12,466,506	1.49
60-66	67,467,007	45,083,184	37,995,389	52,370,886	52,627,119	1.47

Table 3: Performance on the 15-puzzle clustered by cost

8 Solution Cost and Bidirectional MVC

Given the measures here, we then considered whether we could predict if hard problems previously unsolved might be solved with bidirectional search. In particular, Schütt, Döbbelin, and Reinefeld (2013) generated a problem instance for the 24-puzzle with a solution cost lower bounded by 140. Despite having a very strong cluster of computer nodes, they failed to solve this problem instance after running IDA* for three months using the strong 8-8-8 PDB heuristic and expanding $42,854,920,933,846$ nodes.

In our tests on the 24-puzzle and the 15-puzzle with various heuristics we found that M1–M3 all returned small values, suggesting that bidirectional search would not perform well. But, as was suggested in Section 4.4, when the problem instances are extremely difficult to solve and their solution depth is large, such as the mentioned problem instance, additional analysis can be performed.

Korf, Reid, and Edelkamp (2001) observed that a heuristic does not prune nodes in the first few levels of the search tree and only takes effect as the search deepens. As the optimal solution cost grows larger, the number of levels for which a heuristic has no pruning power increases. Thus, while the number of states in the shallow levels of the search will be unchanged as the problem difficulty increases, the number of critical states can continue to grow.

We demonstrate this phenomenon in Table 3 on the 15-puzzle using Korf's set of 100 instances (Korf 1985). The first column of the table is the solution cost, clustered into buckets. The next five columns show the number of necessary expansions resulted by different algorithms using the Manhattan distance heuristic. For unidirectional search algorithms we experimented with forward A*, backward A* and with Best-UNI (which is the best of these on a per instance basis). For bidirectional search we experimented with DVCBS and NBS. Finally, the last column is the ratio between the number of nodes expanded by NBS and the Best-UNI algorithm. As can be seen from the results, this ratio decreases as the solution cost increases. Similar experiments with a 4-5-5-5 PDB heuristic confirm this trend. Thus, even though our measurements suggest that this problem instance is unidirectional, it is still possible that bidirectional search would perform well on this problem due to its overall difficulty and large solution depth. It is a matter of future work to solve this problem and see.

9 Conclusions

Our results suggest that inexpensive measures can be used to predict the performance of bidirectional search algorithms, in particular by looking for critical states in the state space. Future work in this direction will look more deeply into the impact of the problem difficulty, and will also look at how heuristics are constructed in planning, where automated heuristic construction may result in many critical states, as bidirectional search has been successful in this context (Torralba, López, and Borrajo 2016). Finally, it is important to study the choice of heuristics. Current heuristics are optimized for unidirectional search, but different methods for building heuristics may favor bidirectional search.

Acknowledgments

We acknowledge the support of CIFAR and the Natural Sciences and Engineering Research Council of Canada (NSERC). This work was also supported by Israel Science Foundation (ISF) grant #844/17 to Ariel Felner and Eyal Shimony, by BSF grant #2017692, by NSF grant #1815660 and by the Frankel center for CS at BGU.

References

Alcázar, V.; Riddle, P.; and Barley, M. 2020. A unifying view on individual bounds and heuristic inaccuracies in bidirectional search. In *AAAI Conference on Artificial Intelligence*.

Barker, J. K., and Korf, R. E. 2015. Limitations of front-to-end bidirectional heuristic search. In *AAAI*, 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 *IJCAI. Also avaialble at http://arxiv.org/abs/1703.03868*.

Culberson, J., and Schaeffer, J. 1996. Searching with pattern databases. In *Proceedings of the 11th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, volume 1081 of *Lecture Notes in Computer Science*, 402–416. Springer.

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

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

Eckerle, J.; Chen, J.; Sturtevant, N. R.; Zilles, S.; and Holte, R. C. 2017b. Sufficient conditions for node expansion in bidirectional heuristic search. In *ICAPS*.

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

Holte, R. C.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *Artif. Intell.* 170(16-17):1123–1136.

Holte, R. C.; Felner, A.; Sharon, G.; Sturtevant, N. R.; and Chen, J. 2017. Bidirectional search that is guaranteed to meet in the middle. *Artificial Intelligence Journal (AIJ), In press*.

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

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of iterative-deepening-A*. *Artificial Intelligence* 129(1–2):199–218.

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

Lelis, L. H. S.; Stern, R.; and Sturtevant, N. R. 2014. Estimating search tree size with duplicate detection. In Edelkamp, S., and Barták, R., eds., *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014*. AAAI Press.

Schütt, T.; Döbbelin, R.; and Reinefeld, A. 2013. Forward perimeter search with controlled use of memory. In Rossi, F., ed., *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 659–665. IJCAI/AAAI.

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 *SoCS*, 82–90.

Shaham, E.; Felner, A.; Sturtevant, N. R.; and Rosenschein, J. S. 2018. Minimizing node expansions in bidirectional search with consistent heuristics. *Symposium on Combinatorial Search (SoCS)* 81–89.

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

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2):144–148.

Torralba, A.; López, C. L.; and Borrajo, D. 2016. Abstraction heuristics for symbolic bidirectional search. In *IJCAI*, 3272–3278.

Valenzano, R. A., and Yang, D. S. 2017. An analysis and enhancement of the gap heuristic for the pancake puzzle. In *Symposium on Combinatorial Search*, 109–118.