A Comparison of Bi-Directional Search to Unidirectional Least-Cost Path Algorithms

S. Kinahan, S. Jena, P. Kodali, and M. Pandey

Abstract—**We implement a bidirectional search algorithm and evaluate its performance in comparison to breadth-first, depth-first, and A* search in terms of search behavior, algorithmic complexity, and computational time.**

Each least-cost path search algorithm is evaluated in a simplified maze-solving domain. The performance of each of these algorithms is compared across problem environments of varying sizes and complexities.

We attempt to isolate the domain characteristics which have a significant impact on the effectiveness of each algorithm. Based on these results we estimate the relative strengths and weaknesses for each search approach when applied to a given problem case in the maze-solving domain. Finally, we present statistical results to support our analysis.

Index Terms—**Artificial intelligence, Data structures, Intelligent systems**

I. Introduction

Typical path-finding algorithms process nodes in a single direction, working from the start state towards the goal state. Bi-directional search algorithms combine forward and reverse direction searches from the start and goal state simultaneously.

Holte et. al (2016) presents the first bi-directional heuristic search algorithm MM for which it is guaranteed that these search processes will meet in the middle under all circumstances.

Although Holte presents proofs of the properties that guarantee the optimality of solutions returned by MM, a primary conclusion made in the paper is that a more thorough experimental comparison of MM across varied domains remains to be done.

Holte similarly stops short of claiming that MM is the best bidirectional search algorithm in terms of minimizing run time or the total number of nodes expanded. Exploring these unanswered questions is the primary motivation for our exploration of the search behavior and performance of MM. Other bi-directional search algorithms are not implemented nor measured against MM here.

Proving theoretically that the MM algorithm is guaranteed to meet in the middle is outside the scope of this report, this topic is covered fully by Holte (2015).

Comparing the search behaviors of MM to other optimal shortest-path algorithms such as breadth-first search (BFS), depth-first search (DFS), uniform-cost search (UCS) and A* search will allow us to categorize the strengths and weaknesses of each approach. We attempted to identify those conditions under which one search approach will expand a node while another approach will not.

In addition to total nodes expanded, we compared search behaviors based on when a particular node is chosen for expansion during the search process. We will leverage these observations to identify specific conditions under which MM can be expected to perform better or worse than other unidirectional searches on average.

Algorithmic complexity is another metric we used to compare the MM algorithm approach to BFS, DFS, UCS, and A* search in theoretical terms.

To quantify the actual observed performance of each search algorithm, we performed statistical analyses based on features such as the total nodes expanded, path cost, and computational time taken for a given search problem. Data was gathered across domains of varying size and complexity.

Our team hypothesized that the performance of MM when applied to most Pacman search domains would at best approach or closely mirror the performance of A* search. The reasoning behind our hypothesis is that although MM has a unique way of ordering the Open set and therefore selecting the next node chosen for expansion, the overall implementation is based on two A* searches running essentially in parallel. We felt it was unlikely that the MM algorithm would be able to significantly outperform the A* algorithm, given that A* search with an admissible heuristic is complete, optimal, and guaranteed to find an optimal solution.

On the other hand, we expected that MM would significantly outperform BFS and DFS, regardless of the search layout or metric used for comparison. These more simplistic search implementations have several properties which make them weaker competitors to MM. For example, BFS has the property of completeness, meaning it is guaranteed to find a solution if it exists. It does not however guarantee an optimal solution. Holte et. al (2016) claims that MM does find an optimal solution, so the only cases where we would expect MM to exhibit similar performance to BFS would be very small or trivial test domains.

This work was created at Arizona State University Fall 2020 for course CSE 571 under Dr. Yu Zhang.

Kinahan, S. is with Arizona State University, Tempe, AZ 85281 USA. (skinahan@asu.edu)

Jena, S. is with Arizona State University, Tempe, AZ 85281 USA. (sjena@asu.edu)

Kodali, P. is with Arizona State University, Tempe, AZ 85281 USA. (pkodali1@asu.edu)

Pandey, M. is with Arizona State University, Tempe, AZ 85281 USA. (mpandey4@asu.edu)

The primary achievement of this project was the successful integration of the bidirectional search algorithm MM given by Holte et. al into the UC Berkeley Pacman domain. This practical application of Holte's theories allowed us to explore the strengths and weaknesses of MM when applied to a real problem space. Our implementation contains some issues that hamper performance, these are enumerated in the results section along with suggestions for further refinement.

During this project we also generated valuable data on the relative performance of unidirectional DFS, BFS, UCS, and A* search algorithms across several maze layouts in the Pacman domain.

II. Technical Approach

For this project, we implemented the MM algorithm described by Holte in the Pacman domain given by the UC Berkeley AI Project (DeNero). This provided the requisite layout flexibility to develop varied test domains for our statistical analysis.

The MM algorithm is a bidirectional heuristic search. This means that MM simultaneously searches from the start state towards the goal state, and from the goal state towards the start state. The search algorithm applied in each direction is a modified A* search.

Standard A* search orders the nodes on the open list based on the sum of the path or backward cost *g(n)* and the heuristic goal proximity or forward cost *h(n)*. The search proceeds by selecting a node for expansion based on the minimum sum value from the open set at each step. MM on the other hand maintains two open sets, $Open_F$ and $Open_B$. The priority of a given node in the open set(s) is defined as the maximum of $f_x(n)$ and $2g_x(n)$, where x is the search direction Forwards F or Backwards B.

This $2g_x(n)$ expression ensures that for a given node and search direction, nearby nodes are guaranteed to be considered by MM for expansion before more distant nodes. Similarly, it is guaranteed that distant nodes to a forward search are much more likely to be expanded (if at all), by the reverse direction search. On each iteration, MM selects the node with the minimum priority C to expand based on the minimum across both sets:

$C = min(prmin_F, prmin_B)$

$prmin_F = min(Open_F)$

$prmin_B = min(Open_B)$

MM also tracks a value U which represents the lowest-cost solution found by the algorithm so far. This value U is updated whenever a better solution is found. MM will stop searching when U is less than or equal to the largest of C, $fmin_F$, $fmin_B$, and ($gmin_F + gmin_B + \varepsilon$), where ε is the cost of the cheapest edge in the domain. Intuitively we can see that if U is less than or equal to any of these parameters, then we are guaranteed to have found an optimal solution and can stop searching.

Holte et. al (2016) contains complete pseudocode needed to implement the MM algorithm. However, to provide a solution suitable to the UC Berkeley Pacman domain we had to make some modifications. For example, the pseudocode provided by Holte only returns the cost of the optimal solution. Our implementation of the MM algorithm has been modified to return a path of the optimal steps, which are then provided to the Pacman agent for execution.

By default, the UC Berkeley Pacman domain measures computational time in seconds. We found that this limited our ability to get meaningful metrics of computational time, so made alterations to the code base to report these results in milliseconds.

Our implementation of the MM algorithm was created alongside the implementation of A* search in the search.py file given by the UC Berkeley Pacman project.

For the qualitative comparison of search behaviors between BFS, DFS, UCS, A*, and MM algorithms, we used pseudocode available for each algorithm to consider how each approach selects nodes for expansion from the Open set. We used the pseudocode and theoretical definitions of each search algorithm to draw comparisons based on algorithmic complexity.

For the quantitative comparison of MM against BFS, DFS, UCS, and A*, we gathered data on a variety of performance characteristics across diverse search layouts.

The search layouts used in this project were taken from the predefined layouts included in the UC Berkeley AI project Pacman domain. Specifically, the layouts used for data collection were tinyMaze, mediumMaze, bigMaze, and openMaze to compare DFS, BFS, UCS, and A* search performance for standard position search problems.

The tinyCorners, mediumCorners, and trickySearch layouts were also used to gather data on BFS, A* search, UCS, and MM. For each search algorithm and layout pair we gathered metrics for the overall path cost returned by the algorithm, the computational time taken to solve the search problem, the total number of search nodes expanded, and the overall score earned by the Pacman agent.

These metrics were used to generate figures which illustrate the comparative performance of each algorithm when applied to each problem space.

III. Results

Unidirectional search algorithms DFS, BFS, UCS, A* and the Bidirectional MM algorithm implemented in this project were run on three different layouts (tiny maze, medium maze and big maze) for 100 iterations each. The total number of nodes expanded and computational times were obtained for each iteration. Computational time, total number of nodes expanded, and choice of heuristic were the primary features chosen for analysis.

Computational Time:

The average computational time taken to find the path by different search algorithms in these layouts are shown in the following graphs (Fig 1-3).

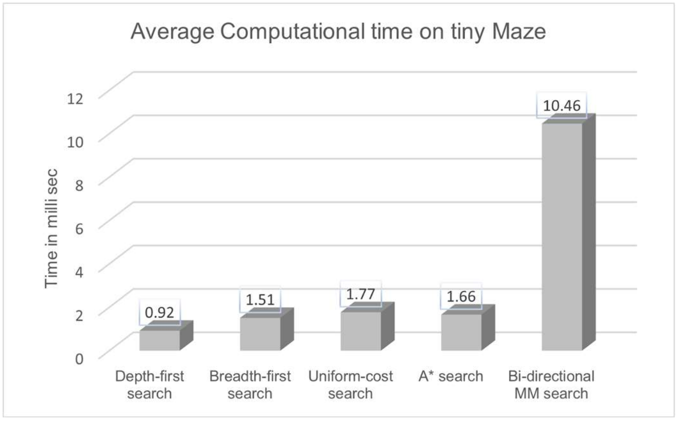

Fig. 1. Average computational time taken in tiny maze layout.

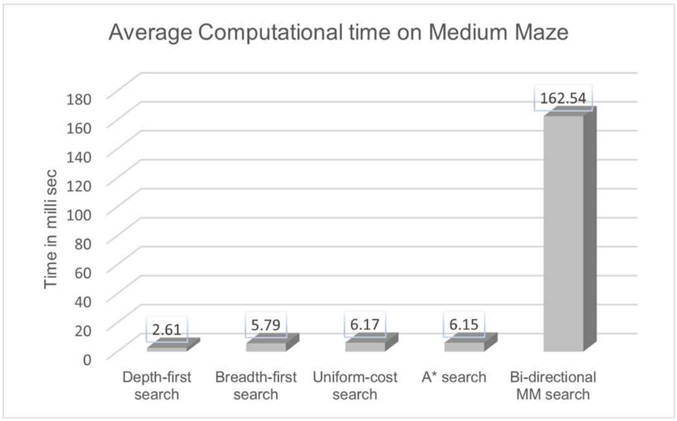

Fig. 2. Average computational time taken in medium maze layout.

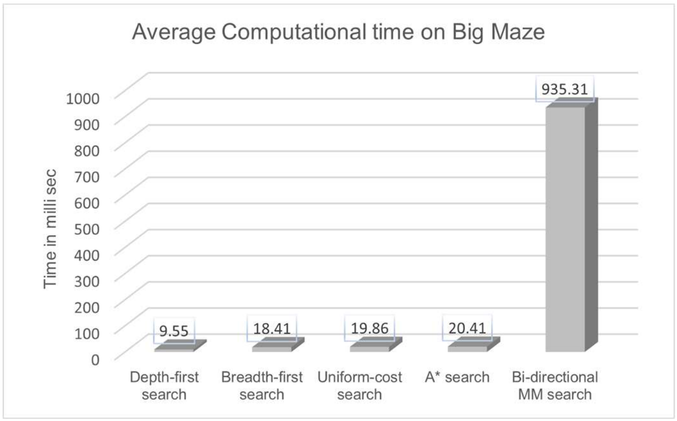

Fig. 3. Average computational time taken in big maze layout.

As demonstrated by Figures 1-3, the average computational time taken by MM search greatly exceeds the time taken by unidirectional search algorithms.

To review, the worst case time complexity of DFS is *O(b^m)*, where *b* is the branching factor and *m* is the depth of the solution found. Breadth-first search has a worst-case time complexity of *O(b^d + 1)*, where *b* is the branching factor and *d* is the level at which the solution is found. UCS has a worst-case time complexity of *O(b^C/e)*, where *C* is the cost of the optimal solution and every action costs at least *e*. The time complexity of A* will depend on the heuristic chosen, but in the worst case A* will have a complexity of *O(b^d)*, where once again *b* represents the branching factor and *d* represents the depth of the solution.

To analyze the worst case time complexity of MM, we must consider the operation of the algorithm. The most important repeated step in the MM algorithm is the selection of the next node for expansion. This selection is done by finding the minimum cost node across the Open sets in the forward and backward direction. The complexity of this operation is given in the following paragraph. However, before this selection can be made, we must first assign a priority to those nodes currently being considered. In our implementation this is done via a costly operation over all nodes in both the Forwards and Backwards Open sets. This is a departure from the MM pseudocode, which assumes that the *fminF, gminF, fminB,* and *gminB* are all readily available. This operation is done at the top of the primary loop of the algorithm and is the most likely culprit for the low efficiency we observed in terms of computational time.

The greater computational time in Bidirectional MM search relative to A* search matches our hypothesis. This is because in Bidirectional search, the min *g(n)* and min *h(n)* must be computed for both the forward and backward direction, even though a priority queue is implemented over both open sets:

F (n) = max(f F (n), 2gF(n))

In a priority queue, extracting the node with highest priority has a time complexity of *O(lg n)*. This holds for a single priority queue whose key values are the cost itself, but that is not the case with our implementation. We have a priority queue for the forwards and backwards opens sets, but the calculation for min *g(n)* and min *f(n)* is being done by traversing the entire array and backtracking from each node to the start or goal state respectively. This yields an overall time complexity of *(n^d + lg n)*, where *d* once again represents the depth of the solution. If we had used a priority queue for *g(n)* and *f(n)* as well, then we would expect the complexity would reduce to *(3 lg n) lg n* for each priority queue.

Total nodes expanded:

As demonstrated in the graph in Fig. 4, a Bidirectional MM search almost always performs better than other unidirectional least-cost search algorithms in our selected experiment layouts. The only algorithm which outperforms MM is DFS, but in this case MM still expands less total nodes in the case of a small maze.

We also found that the performance of DFS varies based on the position of the goal state, whereas MM remains mostly consistent in that regard. The other three unidirectional search algorithms BFS, UCS, and A* expanded an equal number of nodes for the three different maze layouts.

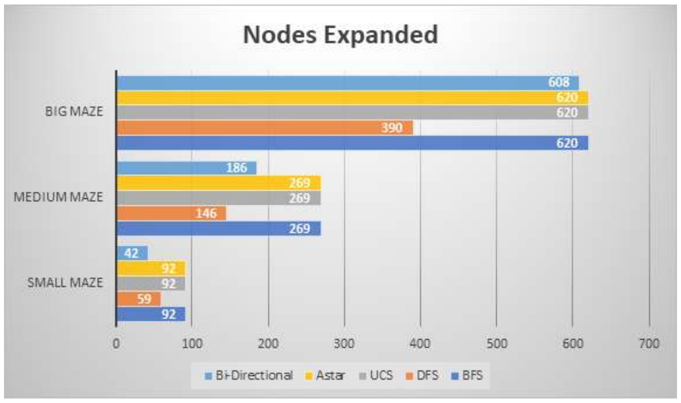

Fig 4. Number of nodes expanded in different algorithms on the small, medium, and big maze. Manhattan heuristic was used for this analysis

Heuristic selection:

To better understand the effect of the chosen heuristic on the performance of these search algorithms, both A* search and Bidirectional MM search were tested using an admissible heuristic function and without a heuristic (null heuristic) in the Position Search Problem and Corners Problem.

In most cases we found that a bidirectional MM search expands less total nodes than an A* search using the same layout. But under specific conditions such as the big maze, or tiny maze using the manhattan heuristic, A* outperforms MM search as illustrated in Fig 5.

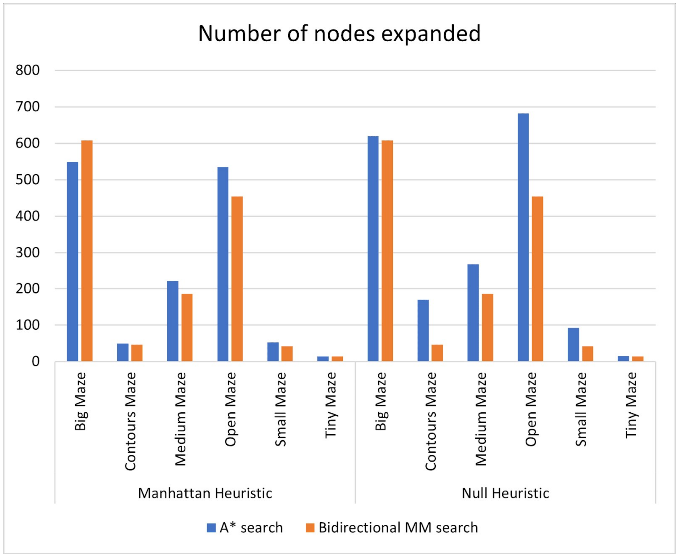

Fig 5. Comparing the number of nodes expanded in different maze layouts between A* and bidirectional searches.

Another important observation from the above experiment is that the number of nodes expanded in bidirectional search in different layouts does not vary when altering the heuristic used in Position Search problems. In Fig. 5, the number of nodes expanded in bidirectional search using manhattan and null heuristics are the same. Across these layouts we can therefore reasonably conclude that the chosen heuristic does not have an effect on the total nodes expanded by MM. This is in fact an expected result based on Holte et. al (2016), because the MM algorithm with $h(s) = 0\ \forall s$ yields the brute-force version of the algorithm MM_0. In the following observations we will not make a distinction between MM and MM_0, except by identifying cases where the null heuristic was applied.

Corners Problem:

Fig. 6 demonstrates that in the Corners Problem, the number of nodes expanded by MM is less than the nodes expanded by A* search. This holds true in all the experimental layouts except for the tiny maze with heuristic.

Unlike in the Position Search problem, a significant difference in the number of nodes expanded with null heuristic and corners heuristic can be noticed in the Corners Problem as shown in Fig 7.

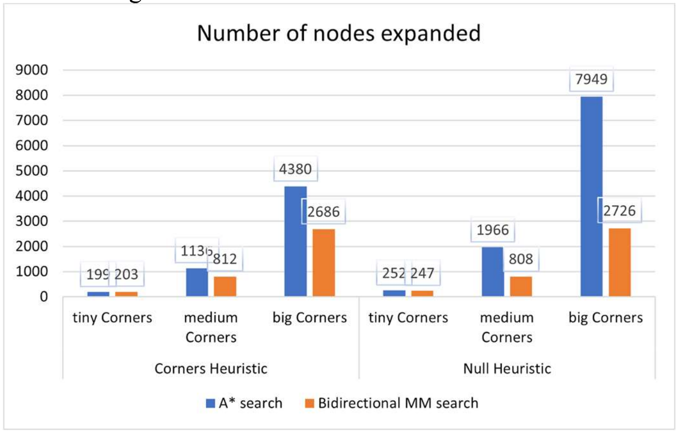

Fig 6. Comparing the number of nodes expanded in different corner layouts in A* and bidirectional searches.

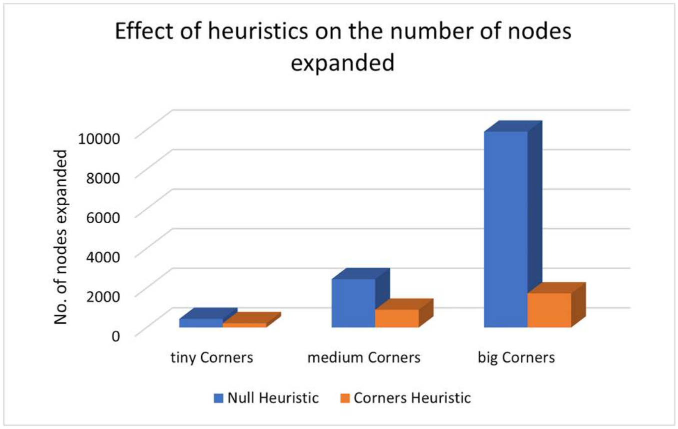

Fig 7. Comparing the nodes expanded in Bidirectional search under null heuristic and with corner heuristic

IV. CONCLUSION

The main objective of search algorithms is to find the optimal or most efficient path from the start state to the goal state. The results obtained from our experiments conducted in the Pacman search domain clearly suggest that the bidirectional MM search performs better on average than the chosen unidirectional search algorithms.

Although the computational time of Bidirectional search exceeds the other search algorithms, this issue could be addressed by implementing additional priority queues to hold the g(n) and f(n) values. This would lead to a tradeoff of greater memory usage in exchange for more efficient calculations of node priority. In some scenarios, A* search is observed to have expanded fewer total nodes than Bidirectional search.

However, multiple runs on different search spaces show that on average the number of nodes expanded by bidirectional search is lower than A* search. Therefore we can reasonably infer that bidirectional search will tend to outperform these unidirectional search algorithms in terms of the number of nodes expanded.

This comparative analysis is limited to the UC Berkeley Pacman domain. For future studies we recommend that the implementation of MM Bidirectional search be extended to more varied game domains such as the Pancake puzzle, Rubik's cube, eight puzzle game, and others. This broader variety of game domains may help to gather insights into the performance of bidirectional search. This could help to identify the domain specific strengths and weaknesses of different search algorithms, for later application in pathfinding problems.

References

Holte R.C.; Felner, A.; Sharon, G.; Sturtevant, N. R. (2016). "Bidirectional Search That Is Guaranteed to Meet in the Middle". presented at AAAI-16. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/download/12320/12109

Holte, R. C.; Felner, A.; Sharon, G.; and Sturtevant, N. R. 2015. Bidirectional search that is guaranteed to meet in the middle: Extended Version. Technical Report TR15-01, Computing Science Department, University of Alberta.

DeNero, J.; Klein, D.; "UC Berkeley Pacman AI Projects" [Online]. Available: http://ai.berkeley.edu/project_overview.html