Bidirectional Search in The Pacman World (April 2020)

Gautam Sharma, Hoda Nasser, Yash Mandlik, Yashaswy Govada

***Abstract*—Shortest Path Finding algorithms are routinely used to address numerous problems, such as vehicle routing, network design and military applications, to name a few. Numerous Artificial Intelligence (AI) algorithms have been developed to find the shortest path between two states, with varying levels of efficiencies in different domains. This report focuses on the implementation of the MM search strategy, a bidirectional heuristic search algorithm, in the Pacman game. The search algorithm will be used to find a path from a start state to a goal state. The report provides a performance comparison between the MM, A*, Breadth First and Depth First Search algorithms within the same application and conditions. The report compares the algorithms based on number of expanded search nodes and whether or not it reaches the goal to select the most suitable algorithm for the Pacman game. The environment's conditions will change based on the size of the map and its complexity defined by the number of walls. Based on the tests results, the bidirectional search MM with maze distance heuristic seems to perform better than other algorithms in terms of nodes expanded, yet it required more computational time. The results of this study will help a user identify the best algorithm and heuristic to use depending on his or her goals and objectives.**

***Keywords* - Shortest path finding problem, artificial intelligence, search algorithms.**

I. INTRODUCTION

Shortest path finding is a recurring problem in many industrial applications, such as IP routing, connecting telephone networks, and navigation applications. It is also widely used in gaming and military applications. Due to its importance, many Artificial Intelligence (AI) based algorithms were designed to optimize shortest path finding for different domains and expected performance, such as least amount of computations or computational speed. Search algorithms are usually classified into two different categories: informed and uninformed search algorithms.

An informed search algorithm is a heuristic search that calculates a heuristic value to estimate how far a given state is from the goal state and utilizes that value to get to the goal state [1]. Two examples of informed search algorithms are MM and A* search algorithms. MM is a bidirectional search algorithm which triggers two separate searches, one from the start state forward to the goal and another from the goal state backward to the start state. The MM algorithm guarantees that both searches will meet in the exact midway of the path to the goal [2]. On the other hand, A* looks for the lowest cost from the start state to the next state in addition to the heuristic value which estimates the lowest cost state to the goal state. At each state, the A* algorithm calculates the sum of the two values to select the cheapest next state until it reaches the goal.

An uninformed search is a blind search mechanism that looks for a path in all directions without any information about the success of the path before it either reaches the goal or a dead end [3]. Two examples are the depth-first and breadth-first search algorithms. The depth-first search explores every state path combination until it reaches its possible end, then backtracks every path if it doesn't reach the goal. The breadth-first search explores multiple paths at once by visiting every state in each path in a layered fashion until it reaches the goal.

A traditional example of a path finding problem would be the Pacman's hunt for fruits in its maze. Pacman can run in different maze sizes, with different obstacles like walls which prohibits it from reaching its target, the fruit. Pacman is expected to use the shortest path towards the fruits, in the fastest manner.

The report will implement the MM search algorithm in Pacman's world, and will compare its performance based on distance and speed to the performances of A*, breadth first and depth first searches. The report answers the following questions: how does the MM search compare to other informed and uninformed search algorithms for the Pacman application? What are its strengths and its weaknesses?

II. TECHNICAL APPROACH

The Pacman game is a maze-concept game where the player controls an avatar, the Pacman, through a maze. The player wins by guiding the Pacman towards a fruit, referred to as the "goal" thereafter, using the shortest path possible to maximize the points won.

The MM algorithm was applied to the Pacman game. A heuristic is specified at the run time and is used by the algorithm to calculate the "cost" of each state, taking into consideration the world state, which is the walls locations in the case of the Pacman. Four heuristics were tested: (1) the null heuristic, (2) the Manhattan distance heuristic, (3) the Euclidean distance heuristic and (4) the maze distance heuristic.

The null heuristic doesn't provide any insights on the distance from the current state to the goal, and only relies on the information of the cost from the original state to the current state. All the other heuristics combine the costs from the origin to the current state and the cost from the current

state to the goal state. The Manhattan distance heuristic is the distance between the current state and the goal state measured along the axis at right angles [4]; the Euclidean distance heuristic is the straight-line distance between current state and the goal [5]; and the maze distance heuristic is the distance from the current state towards the goal taking into consideration all the obstacles in the environment.

The MM algorithm creates two priority queues: the first queue manages the forward search from the start position towards the goal, and the second queue manages the backward search from the goal back towards the start location. The forward queue is initially populated with the start state, and the backward queue is populated by the goal state.

First, the algorithm checks the next state in the forward queue, which is initially the start state. If this state is the goal state, then no other steps are needed, and the path is the collection of the steps in the forward queue. Otherwise, if the state is in the backward queue, it means that the forward queue has met with the backward queue and the path is found. However, if both validations fail, it means that the path has not been found yet and the forward queue needs to explore further states. It is important to note that the algorithm keeps track of every explored state, so each state can only be selected once. The algorithm examines all successor states in case they were unexplored earlier. For each state, the algorithm calculates its cost and adds it to the combined heuristic costs in the forward queue so far.

Similarly, the algorithm repeats the same process with the backwards queue: it selects the next state in the backward queue and examines if it's in the forward queue. If the state is in the forward queue, it means the path from the goal has met the path from the start state and the complete path is found. Otherwise, if the state was not explored earlier, the algorithm examines the successor states. For each state, the algorithm calculates its cost and adds it to the combined heuristic costs in the backward queue so far.

The algorithm then loops through this process: by leveraging the priority queue structure of the forward and backward queues, the algorithm selects the "next" state with the least combined cost and repeats the same process explained above.

The examination of both queues simultaneously at each iteration ensures that both queues are explored equivalently. Once the forward and backward queues meet, the path is formed by combining the actions from the forward queue, along with the reversed actions from the backward queue. Reversing the actions in the backward queue ensures that the actions are assigned in the right order.

III. Results

The MM algorithm was implemented in the Pacman game, and then tested against the other algorithms discussed above on a variety of mazes. For each algorithm-maze combination, four metrics were collected: (1) Game Score, (2) Path Cost, (3) Number of Nodes, and (4) Computing Time.

Fifteen mazes were considered for the testing of the algorithms to provide a variety of challenges and a larger data set for future analysis. The mazes included different sizes, ranging from a 5x5 maze to a 35x35 maze.

On each maze, 10 different approaches were tested: one Depth First algorithm (DFS), one Breadth First algorithm (BFS), four A* algorithms, each with a different heuristic, and four MM algorithms each with a different heuristic. The heuristics considered are: (1) the null heuristic, (2) the Manhattan distance heuristic, (3) the Euclidean distance heuristic, and (4) the maze distance heuristic. All algorithms-heuristics combinations were used in this study. After running all the tests, the data was collected and stored in a spreadsheet format for analysis.

Since the maze is a finite environment and all algorithms are only considering unvisited nodes, all search algorithms are expected to find a path to the goal.

Game Score and Path Cost

The first metrics examined were the Game Score and the Path Cost. Figure 1 shows a plot of all the approaches' scores over the different mazes, and figure 2 shows a plot of all the approaches' path cost over the different mazes. By examining both plots, it is clear that all algorithm almost incur the same cost and receive the same score over all the mazes, except for the DFS algorithm which performed relatively worse in several mazes, especially the "trialMaze1" and the "openMaze", scoring far less and costing far more than all the other algorithms. This indicates that DFS has failed at finding the shortest path from the starting goal to the stated goal in the maze. Due to the DFS logic of inspecting the leftmost branch of the nodes tree till it reaches the goal or the full depth of the tree then backtracking, it would return the left most path to the goal which might not be the shortest in the Pacman environment. Since the path is not the shortest, it would result in a lower score and a higher cost.

Given its divergent performance, DFS was excluded from the remainder analysis and is considered inferior for this application compared to the other algorithms in this study.

Figure 1. Score of every algorithm in different test cases

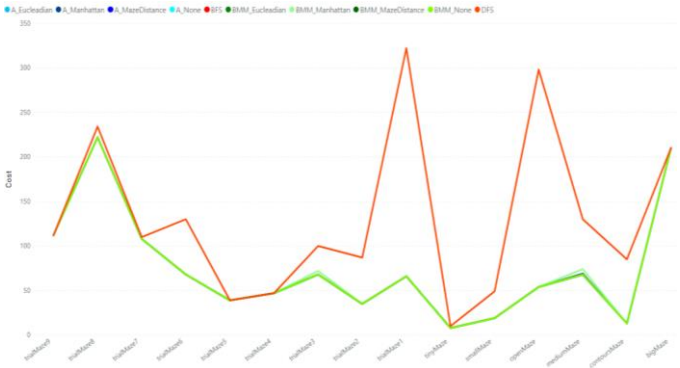

Figure 2. Cost of every algorithm in different test cases

Nodes Expanded

The most important metric for the Pacman game is the number of nodes expanded, as it reflects how efficient an algorithm is in identifying the shortest path to the goal. As the maze grows in size, the number of possible nodes grows and if an algorithm explores more nodes than needed, it might lead to inefficiencies related to memory since it needs to keep tracking previously visited nodes, and computational time, since time would be wasted over the expansion of irrelevant nodes. Table 1 presents descriptive statistics related to the number of nodes expanded by each search approach.

	Mean	Standard Deviation
BFS	323.0667	247.82553
A*- Null heuristic	323.2000	248.03289
A*- Manhattan heuristic	324.6667	247.22074
A*- Euclidean heuristic	325.8000	249.91719
A*- Maze Distance heuristic	323.0667	247.82553
MM - None heuristic	238.7333	196.98349
MM - Manhattan heuristic	224.0000	203.07423
MM - Euclidean heuristic	218.0000	198.53607
MM - Maze Distance heuristic	125.9333	96.51903

Table 1. Mean and Standard deviation of the number of nodes expanded by each search algorithm

To identify whether any of the algorithms is significantly different from the others, a Repeated Measures Anova test was conducted. Table 2 summarizes the Anova findings.

Source	Type III Sum of Squares	df	Mean Square	F	p-value
Intercept	10569820	1	10569820	23.94	.000237
Error	6180104	14	441436.055		

Table 2. Summary of Repeated ANOVA results

The reported p-value = 0.00023 < 0.05 indicates that the test results are significant, meaning there are at least two algorithms that are statistically significantly different in terms of nodes expanded (p = 0.0002< 0.05) from one-another.

To further identify which algorithms are significantly different from one another in terms of nodes expanded, an LSD post-hoc test was conducted. Every MM algorithm (with every heuristic tested) expanded significantly less nodes than all other algorithms tested, regardless of which heuristic was used. Moreover, the MM algorithm with the maze distance heuristic expanded statistically significantly less nodes than any other algorithm, including other MM approaches with different heuristics. Table 3 summarizes the results of the LSD post-hoc tests as it relates to the MM with maze distance heuristic.

Initial Algorithm	Compared to	Mean Difference	Significance
MM - Maze distance heuristic	BFS	-197.133	.000
	A*- Null heuristic	-197.267	.000
	A*-Manhattan heuristic	-198.733	.000
	A*-Euclidean heuristic	-199.867	.000
	A*-Maze distance heuristic	-197.133	.000
	MM-Null heuristic	-112.800	.001
	MM-Manhattan heuristic	-98.067	.005
	MM-Euclidean heuristic	-92.067	.006

Table 3. LSD Post-hoc Pairwise Comparison

On average, the MM algorithm with the maze distance heuristic expanded between about 200 nodes less than the other algorithms, and about 100 nodes less than MM algorithms with other heuristics, and the differences are significant at the 95% confidence level.

By further examining the maze distance heuristic, it is also noticeable that within the A* tests, the A* search with the maze distance heuristic performed slightly, but not significantly, better than the remaining A* with different heuristics. This further emphasizes the reduction in number of nodes expanded achieved by selecting the maze distance heuristic. To further study this heuristic, the computation time metrics analysis was conducted as direct comparison between the MM algorithm with maze distance heuristic and the A* algorithm with maze distance heuristic.

Computation Time

Figure 3 shows the computation time of the different approaches over the different test mazes. Though the computation time may vary over different systems, it is noticeable that the MM and A* search, both with maze distance, required more computational time than the remaining algorithms.

The results are both logical and intuitive: the heuristic calculation would realistically require more computation compared to other uninformed algorithms, such as DFS and BFS; and the tests were conducted in a limited maze space, which means a reduced amount of possible expandable nodes.

It is expected that on a large-scale maze, this computational time will be less significant compared to other algorithms that would expand more needless nodes.

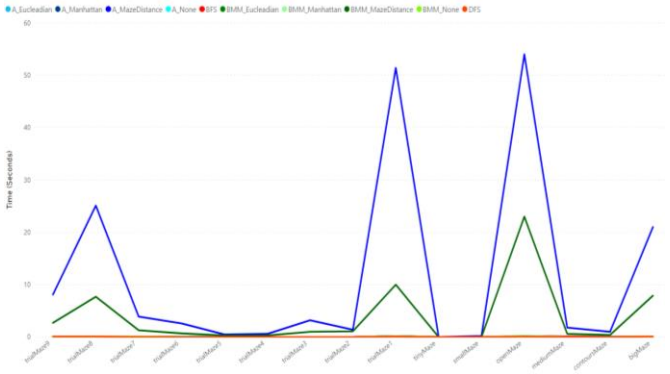

Figure 3. Time spent by every algorithm in different test cases

To directly compare the computation time of the A* algorithm with the maze heuristic and the MM algorithm with the maze heuristic, a paired t-test was conducted. Table 4 summarizes the test results.

	Compared Algorithms	Mean	Significance (2-tailed)
Paired Samples Test	MM with maze distance – A* with maze distance	7.85333	0.032

Table 4. Paired Samples Test Results

The MM search when using the maze distance heuristic was significantly faster than the A* with maze distance heuristic search, and the difference is significant at the 95% confidence level ($p=0.032<0.05$). On average, the MM algorithm was 7.8 seconds faster than A*.

IV. Conclusion

All algorithms, except depth first search, consistently achieve the goal of retrieving the shortest path which results in the highest score for the Pacman game. Yet, the MM search with maze distance heuristic has achieved this goal while expanding a lower number of nodes than all other tested approaches. Furthermore, the heuristic choice has a major impact on the algorithm performance, as the MM search with maze distance heuristic significantly outperformed all other MM searches, and the A* search with maze distance heuristic was slightly faster than the other A* approaches. While both algorithms required more computational time than other algorithms when using the maze distance heuristic, the MM search was still significantly faster than the A* algorithm while using the maze distance.

It is very important in algorithm selection, to take into consideration the environment setup. In some cases, based on the size of the environment, it might be better to expand more nodes and use a less efficient algorithm than using a smarter algorithm with good heuristic to get results faster. The results of this study will help a user make an informed decision regarding which algorithm and heuristic to use depending on the user's goals and objectives.

V. Team Effectiveness

The team has collaborated throughout the whole project, tasks were split equally between team members and weekly meetings were held to monitor progress and tackle any challenges faced.

References

[1] "Informed Search Algorithms in AI - Javatpoint." https://www.javatpoint.com/ai-informed-search-algorithms (accessed Apr. 30, 2020).

[2] R. C. Holte, A. Felner, G. Sharon, N. R. Sturtevant, and J. Chen, "MM: A bidirectional search algorithm that is guaranteed to meet in the middle," *Artif. Intell.*, vol. 252, pp. 232–266, 2017, doi: 10.1016/j.artint.2017.05.004.

[3] "Uninformed Search Algorithms - Javatpoint." https://www.javatpoint.com/ai-uninformed-search-algorithms (accessed Apr. 30, 2020).

[4] P. E. Black, "Manhattan distance," *Dictionary of Algorithms and Data Structures*, 2019. https://www.nist.gov/dads/HTML/manhattanDistance.html.

[5] P. E. Black, "Euclidean distance," *Dictionary of Algorithms and Data Structures*, 2004. https://www.nist.gov/dads/HTML/euclidndstnc.html.