CSE 571 Final Project Report : Bidirectional Search that is guaranteed to meet in the middle

Sandeep Kallepalli skallep2@asu.edu 1225463655
Appari Lalith alalith@asu.edu 1225493945,
Aman Peshin apeshin@asu.edu 1225476655,
Vishal Jaimin Vakil vvakil1@asu.edu 1225551353

***Abstract*—Bidirectional search is an exciting new search algorithm devised by Robert et al. where they implement a forward search from the start node and a backward search from the goal which is always guaranteed to meet at a point in the middle. In this paper, we implement this algorithm as well as other algorithms like A*, Uniform Cost Search, Breadth First Search, and Depth First Search and compare their performance in the Pacman domain provided by CS188. We will be devising new and improved Pacman of varying size, complexity and different start, goal states. In conclusion, we would be providing scenarios where bidirectional search which is extremely powerful to find optimal paths through the mazes provided as compared to other search algorithms.**

***Index Terms*—BiDirectional Search, A* Star, Meet in Middle, Heuristics,**

I. Introduction

BIDIRECTIONAL search algorithms are proven to be effective in many cases as compared to the unidirectional search algorithm. Bidirectional search starts by initializing two lists; one for the path from the start to the goal and one for the path from the goal to the start. Each iteration the algorithm expands the search from both directions, meaning the search from both ends progresses at the same time. It is believed to be one of the most efficient methods of searching because it only visits a node once and is guaranteed to find the optimal solution. Therefore, bidirectional search is a good choice when looking for the shortest path between two nodes. It is an algorithm which is known for its quickness, being more efficient than a unidirectional search, as well as being guaranteed to find the optimal solution.

In this project, we implemented the ”Meet in the Middle” (MM) algorithm with and without heuristics to compare the results with other implemented algorithms such as Depth First Search, Breadth First Search, Uniform Cost Search and A*. The MM and MM0 algorithm works like an A* algorithm but has two separate search paths in it. There is a forward and backward search in MM and MM0 algorithm which meets in the middle meaning, the forward and backward search algorithms would not expand the nodes whose g-value is more than the half of the optimal solution cost. The forward search starts from the initial state while the backward search starts from the goal state.

Various other comparisons were also made like adding the heuristics to algorithms with null heuristics. It was also noted that the algorithms with heuristics that are not null expanded lesser nodes than the algorithms with the null heuristics. It can be found that this algorithm always opens the nodes that are near to the goal. If the distance cost between the node and goal is less than or equal to the (optimal solution cost)/2 then and only then the MM or MM0 algorithm will expand that node otherwise it won't expand. Doing so will yield perfect results in terms of pathcost. The distance between the nodes for MM or MM0 algorithms and start or goal state for forward or backward search, is defined by classifying it into 3 categories viz near, far and remote. If the distance cost between the nodes and the start state or goal state is less than or equal to the (optimal solution cost)/2 i.e. if they are expandable then those nodes are said to classified under the near category while if the distance cost between the nodes and the start state or goal state is more than the (optimal solution cost)/2 and less than or equal to the optimal solution cost then those nodes are classified under the “far” category. Additionally if the distance cost between the nodes and the start state or goal state is more than the optimal solution cost, then those nodes are classified as “remote”. Through this classification the distance of the node from the start state and goal state can be categorized to determine if it is expandable or not. Let's say a node is denoted by near-far (NF) then the node is closer to the start state but far from the goal state.

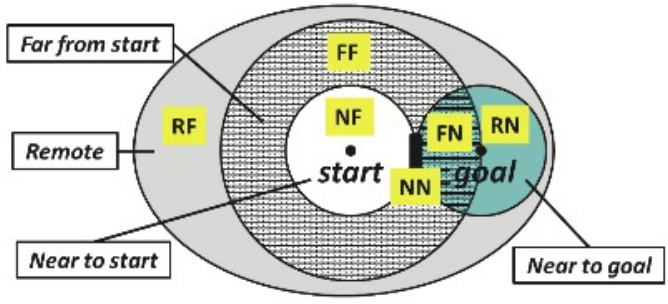

Fig. 1. Path cost for different search algorithms

However, we conclude that the MM and MM0 algorithm are not guaranteed to expand fewer nodes or less run time than the other algorithms. In this project we implemented the bidirectional search algorithms for pacman path-finding problems.

To summarize the algorithms:

- MM forward or backward search never expands to a node if it is classified as remote or far from the initial (start) or goal state.
- MM never expands a node if its f-value exceeds optimal solution cost.
- MM always returns the optimal solution cost.

Algorithm 1 PsuedoCode for MM

```
openf.push(startstate)
openb.push(goalstate)
gfs(startstate) = 0 , gbs(goalstate) = 0
U ← inf
While Open_f ≠φ & Open_b ≠φ
    Get the top node in both the open sets
    C = min(Pr_min_f, Pr_min_b)
    If U ≤ max(C, min(fmin_f, fmin_b, gmin_b+gmin_f))
      Return Path_f + (Path_b)^-1
    End
    If C == Pr_min_f
      Remove node from Open_f and push to Closed_f
      For each child c of node:
        If c in Open_f ∪ Open_b & g_f(c) ≤ g_f(n)+cost(n,c)
          continue
        End
        If c in Open_f
          remove c from Open_f
        End
        If c in Open_b
          remove c from Open_b
        End
        g_f(c) ≤ g_f(n) + cost(n,c)
        Open_f.push(c)
        If c in Open_b
          U = min(U, g_f(c) + g_f(n))
        End
      End
    End
    If C == Pr_min_b
      Repeat the same for backward search
    End
    Return
End
```

II. Technical Approach

We have implemented the bidirectional search in "Bidirectional Search That Is Guaranteed to Meet in the Middle". The search aims to find the optimal cost path between (start, goal) state pairs.

We will be having two searches simultaneously one called forward search(starting from the start state) and the other called Backward search(starting from the goal state). Each search will have a set of search variables like f(g+h), g, open set & closed set. Open set for both the searches will be a priority queue to enable extracting the best node based on the below priority.

$$pr = max(f, 2 * g).$$

Variables For Forward Search : $f_f, g_f, open_f, closed_f$
Variables For Backward Search : $f_b, g_b, open_b, closed_b$
For MM search:

$$f = g + h$$
$$pr_f = max(f_f, 2 * g_f)$$
$$pr_b = max(f_b, 2 * g_b)$$

For MM0 search : We will use Null Heuristic.

$$h = 0$$
$$f = g$$
$$pr_f = max(f_f, 2 * g_f)$$
$$pr_b = max(f_b, 2 * g_b)$$
$$pr_f = 2 * g_f$$
$$pr_b = 2 * g_b$$

Terminal condition for checking the optimal path:

$$U \leq max(C, min(fmin_f, fmin_b, gmin_b + gmin_f))$$

This condition checks that neither the forward search nor the backward search expands an node with distance from starting node is greater than C* and also never expands a node with f-value(g+h) greater than C.

We have implemented a few functionalities in the priority queue.

- isStartPresentInPQ(state) : Returns a boolean stating whether the Priority Queue contains the state node or not
- deletePQ(state) : Void function to delete a state from the Priority Queue.

For addressing the corners problem, we need different successor function for backward search. We have defined a goal state in corners problem and implemented a getSuccessorBS(state) function that gives successor states in backward search.

We have implement the code using the psuedo code mentioned in Algorithm 1 for both MM & MM0.

III. Results, Analysis and Discussions

Comparison of Node expanded and path cost encountered for Default Mazes:

In this section, we are going to compare how Bi-Directional search performs against other search techniques in the inbuilt mazes provided. In small maze and medium maze we see that bi-directional performs better than UCS and BFS, but its results are comparable to that of bi-directional without heuristics, A* and DFS.

In big maze, there is a slightly better for MM0 over

MM since brute force finds an efficient path early, and the difference between the numbers of nodes expanded is very small, and the path cost is the same for all algorithms.

In tiny corners, we see that again A* and Bi-directional have comparable results, but DFS outperforms them all, but at the same time the path cost encountered by DFS is the highest. This is a fair result since we know that DFS will reach a deeper goal state faster albeit with a higher cost. In medium corners, we can see that Bi-directional performs better than A* for both the Manhattan heuristic and our derived heuristic. We can again see that DFS finds the solution with a minimum number of nodes expanded but with a higher cost path.

The same results can be observed on big corners, but the real strength of Bi-directional can be seen over A*, we can see these results when there exist multiple paths from start to goal state.

Maze Type	Bi-Directional MM	Bi-Directional MM0	A*	UCS	BFS	DFS
Small Maze	45	54	53	92	92	59
Medium Maze	177	185	221	269	269	146
Big Maze	610	608	549	620	620	390
Tiny Corners	205	210	199	252	252	51
Medium Corners	797	818	1136	1966	1966	371
Big Corners	2668	2705	4380	7949	7949	504

Fig. 2. Number of nodes expanded for inbuilt mazes

Maze Type	Bi-Directional MM	Bi-Directional MM0	A*	UCS	BFS	DFS
Small Maze	19	19	19	19	19	49
Medium Maze	68	68	68	68	68	130
Big Maze	210	210	210	210	210	210
Tiny Corners	33	33	28	28	28	47
Medium Corners	106	106	106	106	106	221
Big Corners	162	162	162	162	162	302

Fig. 3. Patch Cost for inbuilt mazes

Comparison of Node expanded and path cost encountered for Custom Mazes:

To check our algorithm in different complexities and starting points, we have created 13 derived versions of bigMaze which have multiple paths from start to goal state by removing and creating walls. We have also created a new layout Mega maze which is almost four times the size of a big maze to test our algorithm.

In bdMaze1 to bdMaze4 we see that Bi-directional expands a lesser number of nodes than Bi-directional without heuristics and A* and their path cost is exactly the same. This may be the case since bdMaze all have the same start point which is near the start and far from the goal. But this is changed in bigMazes where we change the starting position of Pacman, here we can see mixed results where A* is obtaining a lesser number of nodes expanded. We have exempted comparing Bi-directional to any non-heuristic search techniques since the number of nodes expanded is going to be higher for them. This is not the case with bigMaze7 where DFS finds a path with higher cost but expands the least number of nodes.

We can see a mixed result in megaMaze where A* start performs better, but on a closer look we have observed that since there were numerous paths from start to goal state and the position of Pacman was in diagonally opposite corners, Bi-directional was able to find a path with lower cost as compared to A*, but in order to do that it expanded more number of nodes. Path cost here plays a very big role due to multiple paths Pacman can take.

Maze Type	Start Node	Goal Node	Bi-Directional MM	Bi-Directional MM0	A*	UCS	BFS	DFS
bdMaze	(35, 1)	(1, 1)	233	222	244	375	375	393
bdMaze1	(35, 1)	(1, 1)	600	598	539	610	610	388
bdMaze2	(35, 1)	(1, 1)	524	537	586	633	633	335
bdMaze3	(35, 1)	(1, 1)	303	301	300	612	612	335
bdMaze4	(33, 19)	(1, 1)	282	274	312	582	582	325
bigMaze	(35, 1)	(1, 1)	610	608	549	620	620	568
bigMaze1	(33, 27)	(1, 1)	315	333	388	666	666	390
bigMaze2	(29,33)	(1, 1)	242	249	278	658	658	437
bigMaze3	(18, 27)	(1, 1)	362	362	315	431	431	466
bigMaze4	(33, 27)	(1, 1)	273	277	316	628	628	355
bigMaze5	(19, 35)	(1, 1)	263	279	313	466	466	267
bigMaze6	(8, 1)	(1, 1)	186	194	195	350	350	424
bigMaze7	(32, 27)	(1, 1)	531	539	497	623	623	283
megaMaze	(68, 70)	(1, 1)	1922	1934	1154	2639	2639	2331

Fig. 4. Number of nodes expanded for Custom mazes

Maze Type	Start Node	Goal Node	Bi-Directional MM	Bi-Directional MM0	A*	UCS	BFS	DFS
bdMaze	(35, 1)	(1, 1)	74	74	74	74	74	214
bdMaze1	(35, 1)	(1, 1)	210	210	210	210	210	210
bdMaze2	(35, 1)	(1, 1)	146	146	146	146	146	210
bdMaze3	(35, 1)	(1, 1)	74	74	74	74	74	210
bdMaze4	(33, 19)	(1, 1)	80	80	80	80	80	180
bigMaze	(35, 1)	(1, 1)	210	210	210	210	210	80
bigMaze1	(33, 27)	(1, 1)	70	70	70	70	70	210
bigMaze2	(29,33)	(1, 1)	68	68	68	68	68	82
bigMaze3	(18, 27)	(1, 1)	77	77	77	77	77	110
bigMaze4	(33, 27)	(1, 1)	70	70	70	70	70	77
bigMaze5	(19, 35)	(1, 1)	76	76	76	76	76	74
bigMaze6	(8, 1)	(1, 1)	57	57	57	57	57	205
bigMaze7	(32, 27)	(1, 1)	113	113	113	113	113	385
megaMaze	(68, 70)	(1, 1)	148	148	362	148	148	32

Fig. 5. Patch Cost for custom mazes

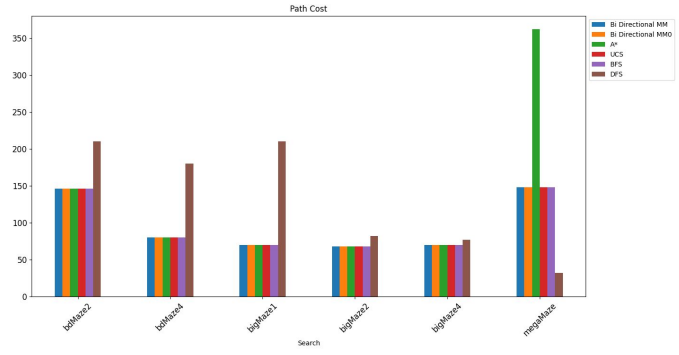

Fig. 6. Path cost for different search algorithms

Bi-Directional MM0	A*	UCS	BFS	DFS
0.005416	0.243325	0.035989	0.035989	0.164165

Fig. 7. T-Test values

Comparing T-Test values for search Algorithms:

We have calculated the T-test values for the above-given algorithms using excel and have tabulated the results, we are comparing Bi-Directional against Bi-Directional without heuristics, A*, UCS, BFS, and DFS. We are going to compare them at a confidence level of 95%. For every algorithm against Bi-directional, we would create a hypothesis that the mean number of nodes expanded is statistically different.

We would be comparing the values against 0.05 and if the value is lesser than 0.05. We can say that the proposed hypothesis is disproved and there is a significant difference between the algorithms in terms of performance. From the table, we can say that Bi-Directional search without heuristics, UCS, and BFS all perform slightly worse than Bi-Directional search since their values 0.005, 0.03, and 0.035 are all lower than 0.05. In the case of A* we see that we have obtained a t value of 0.24 which is significantly higher than that of 0.05, hence at the confidence level of 95% we would say that both of them would expand the almost equal number of nodes, but this does not mean that the algorithm is worse than A*, in this case the deciding factor of choosing an algorithm would be the path cost encountered. Similarly in the case of DFS also, we can say that both DFS and A* at some point have outperformed Bi-Directional but based on our data we see that this was at the expense of high path cost.

IV. Conclusion

We implemented MM, MM0 . We compared the performance of these algorithms with standard uninformed seach methods like DFS, BFS, UCS and with informed searches like A* in different environments. MM and A* were majorly compared and the observation we found is that the MM and MM0 algorithms perform better than A* majority of the times but not always. Finally MM might run in O($b^{(n/2)}$) rather than O(b^n), in the best-case scenario. (Where n is the number of nodes expanded by A* in a unidirectional A* and b is the branching factor). But, the performance of MM depends on how readily the two searches intersect. If they intersect early in the search, a lot of time is saved, but if they diverge drastically, then MM would takes longer than just an A*.

References

[1] *Bidirectional Search That Is Guaranteed to Meet in the Middle*. Holte, R., Felner, A., Sharon, G., Sturtevant, N. (2016). Proceedings of the AAAI Conference on Artificial Intelligence. Available: https://doi.org/10.1609/aaai.v30i1.10436