CSE 571 Final Project Report : Bidirectional Search

Jaehyuk Choi jchoi154@asu.edu 1215326372
Jinyung Hong jhong53@asu.edu 1215173661
Eunsom Jeon ejeon6@asu.edu 1217249085
Yongbaek Cho ycho56@asu.edu 1218501245

***Abstract*—Bidirectional search algorithms includes two separate searches. The basic theorem for this search is that forward search from the start state and backward search from the goal state meet in the middle. However, previous bidirectional heuristic searches do not guarantee to meet in the middle all the time. Since such an issue has arisen, variants of MM were developed for forward/backward search to meet in the middle.**

Inspired by this, we apply the variants of MM including $\texttt{MM}_\epsilon$ and Factional MM (fMM(p)) to solve the POSITION SEARCH problems in Pacman domain. $\texttt{MM}_\epsilon$ is an enhanced version of MM to cover the case that an optimal path is yet found and fMM(p) is based on MM, but more generally used with fractional changes of forward/backward search-space size.

We compare the performances between variants of MM and conventional search algorithms: i) the performances between variants of MM and uninformed search (BFS, DFS) ii) the performances between variants of MM and informed search (A$\star$) using various heuristic functions. The experimental results show the efficiency and effectiveness of each method for path-finding based on the search nodes expanded and processing time. We analyzed how a strong/weak heuristic affects behavior of each search algorithm and concluded that MM outperforms A* most times. however, a strong heuristic might have a trade-off in execution time.

I. INTRODUCTION

By knowing two states: start state and goal state, a bidirectional search algorithm searches concurrently in both directions, interleaving two separate searches. One propagates a normal search forward from the start, and the other one propagates a search backward (i.e. using reverse operators) from the goal. That is, a bidirectional search uses two smaller sub search-spaces rather than a single search-space. When those two sub search-spaces intersect, the algorithm finds an optimal path within the explored search-space and its search process terminates.

The main assumption made by Barker and Korf is that a bidirectional search never expand a node whose cost value (either g_F in forward search-space or g_B in backward search-space) exceeds $C^*/2$, where C^* is the optimal solution cost [1]. From this assumption, we bestow the property such that it "meets in the middle" (MM) and MM is the first bidirectional heuristic search algorithm guaranteed this property [2]. This is achieved by MM's novel selection criterion and priority functions. Based on MM, various extensions of MM have been studied; fMM(p) is a generalization of MM, which uses the priority on paths [3], fractionally changing forward/backward search-space size. $\texttt{MM}_\epsilon$ is an enhanced version of MM that introduces ϵ to the basic priority function [4] in the case that an optimal path is yet found.

Inspired by this, we applied variant versions of MM to solve the POSITION SEARCH problems in Pacman domain for which the Pacman agent should find the shortest path to a food location (goal location).

We compared the performances between MM and conventional search algorithms: i) uninformed search comparison ($\texttt{MM}_0$, $\texttt{MM}_\epsilon$, or fMM vs. BFS or DFS) ii) informed search comparison (MM vs. A$\star$ with various heuristics). The experimental results in Table II and III show expanded nodes and execution time for efficiency and effectiveness of each path-finding method.

II. TECHNICAL APPROACH

In this section, we explain how to implement the variant versions of MM. The instance of a POSITION SEARCH problem is a pair of start and goal states $(start, goal)$ in a state-space in which all edge weights are non-negative. The aim of search is to find a *least-cost* path from start to goal. $C^* = d(start, goal)$ is the cost of an optimal solution where $d(u, v)$ is defined as distance (a least-cost on a path) from state u to state v. We use the usual notation—$f, g, open$, etc.—and use g_min and f_min for the minimum $g-$ and $f-$value on $open$. We have separate copies of these variables for the two search directions, with a subscript (F or B) indicating the direction:

- Forward Search: $f_F, g_F, h_F, open_F, closed_F$, etc.
- Backward Search: $f_B, g_B, h_B, open_B, closed_B$, etc.

The Algorithm 1 shows how we combined MM, $\texttt{MM}_0$, $\texttt{MM}_\epsilon$, and fMM for the implementation of bidirectional search. Line 7 in the Algorithm 1 determines which method among the variants of MM is going to be used.

A. MM

MM guarantees to "meet in a middle" by defining novel selection criterion and priority functions [2], $pr_F^{\texttt{MM}}(n)$ and $pr_B^{\texttt{MM}}(n)$ for forward and backwards, respectively:

$$\begin{aligned} pr_F^{\texttt{MM}}(n) &= \max(g_F(n) + h_F(n), 2g_F(n)) \\ pr_B^{\texttt{MM}}(n) &= \max(g_B(n) + h_B(n), 2g_B(n)) \end{aligned} \tag{1}$$

Furthermore, its stopping condition combines the g_min and the f_min stopping condition functions. We implement MM based on the pseudo-code in [2].

Algorithm 1: Pseudo-code for Combined MM

if *use*$\mathtt{MM}_\epsilon$*==True* **then** use $\mathtt{MM}_\epsilon$ **else** no use $\mathtt{MM}_\epsilon$
if *use*`fMM`*==True* **then** set ratio p **else** p=0.5 for $\mathtt{MM}_0$
$g_F(start) = 0$; $g_B(goal) = 0$; $U = \infty$
$open_F = start$; $open_B = goal$;
Initialize empty set $\emptyset$ for $closed_F$, $closed_B$;
while $(open_F \neq \emptyset)$ *and* $(open_B \neq \emptyset)$ **do**
  $C = min(pr\_min_F, pr\_min_B)$ where $pr\_min_{F/B}$ finds the minimum priority pr_min from taking open set $open_{F/B}$, g-value $g_{F/B}$, and direction F/B
    `/* pr_min is based on conditions (MM, MM_ε, MM_0, fMM) */`
  if $U \leq max(C, f\_min_F, f\_min_B, g\_min_F + g\_min_B + e)$ **then**
    Choose intersect nodes n between $open_F$ and $open_B$;
    Assign the selected n to $state$ and get actions from intersect to start for forward/backward search;
    return $solution$ of the direction for the next move;
  end
  if $C == pr\_min_F$ **then**
    Choose the node from $open_F$ whose $pr_F(n) = pr\_min_F$;
    Expand the forward search-space by taking the node from $open_F$ and adding it to $closed_F$;
    for *each child c of the node* **do**
      Determine which MM function to choose.
      if *c in* $open_F \cup closed_F$ *and* $g_F(c) \leq g_F(n) + cost(n, c)$ **then**
        continue
      end
      if *c in* $open_F \cup closed_F$ **then**
        remove c from $open_F \cup closed_F$
      end
      $g_F(c) = g_F(n) + cost(n, c)$;
      add c to $open_F$;
      if *c in* $open_B$ **then**
        $U = min(U, g_F(c) + g_B(c)$
        `/* update U */`
      end
    end
  else
    // Same process for the backward search;
  end
end
return ∞

B. $\mathtt{MM}_0$

$\mathtt{MM}_0$ is the uninformed (i.e. $h(n) = 0\ \forall n$) version of MM [2]. Without heuristic value, the evaluation function for $\mathtt{MM}_0$ is the following equation:

$$\begin{aligned} pr_F^{\mathtt{MM}_0}(n) &= 2g_F(n) \\ pr_B^{\mathtt{MM}_0}(n) &= 2g_B(n) \end{aligned} \tag{2}$$

That is, while expanding, $\mathtt{MM}_0$ selects nodes from the open list with the smallest cost value (g-value) until it meets the stopping condition.

C. $\mathtt{MM}_\epsilon$

$\mathtt{MM}_\epsilon$ covers the case that an optimal path is yet found. In basic MM, suppose that node n_i in $open_F$ with its cost value $g_F(n_i) = d(start, n_i)$ in a forward optimal path and node n_j in $open_B$ with its cost value $g_B(n_j) = d(n_j, goal)$ in a backward optimal path where node n_i and n_i are in the open set of each search-space for agent to explore in the future [4]. Since the optimal path is yet found, there shall be a gap between node n_i and n_j. That means there exists at least one or more edges from each optimal path that connected node n_i to n_i that basic MM has not explored. The property to cover such a gap leads to the following equations:

$$\begin{aligned} pr_F^{\mathtt{MM}_\epsilon}(n) &= \max(g_F(n) + h_F(n), 2g_F(n) + \epsilon) \\ pr_B^{\mathtt{MM}_\epsilon}(n) &= \max(g_B(n) + h_B(n), 2g_B(n) + \epsilon) \end{aligned} \tag{3}$$

D. fMM

fMM is more generalized version of MM such that it meets at any fraction of the optimal solution cost [3]. Fractionally, the estimation function is calculated as following:

$$\begin{aligned} pr_F^{\mathtt{fMM}}(n) &= \max(g_F(n) + h_F(n), 2g_F(n)/p) \\ pr_B^{\mathtt{fMM}}(n) &= \max(g_B(n) + h_B(n), 2g_B(n)/(1-p)) \end{aligned} \tag{4}$$

where $0 < p < 1$.

fMM's forward and backward searches meet at $(pC^*, (1-p)C^*)$, depending on fractional factor p. In another perspective on fMM, basic MM is a special case of fMM for $p = 1/2$.

E. Heuristics

For informed search, the total performance including the number of expanded nodes or processing time can be influenced by the heuristic function [5]. To see how algorithms' behavior with a weak/strong heuristic, we used various heuristics and the representative heuristics that show effective performances are Manhattan, Octile, Cosine similarity, Diagonal, and Maze distances. The customized heuristic was designed based on the weighted sum of Diagonal distance and fractional minimum/maximum distances between x and y coordinates. Its admissibility and consistency were checked to see clearly the influence of strong/weak heuristics on a search algorithm's behavior. As shown in Table 1, formulas of heuristic functions are illustrated. Additionally, Maze heuristic, which calculates the actual distance using BFS, represents the best performance on both A* and Bi-directional search. This finds the nearest node to add it to the closed set and iterates the process until all nodes have been added into the closed set.

TABLE I: Heuristic Function and Formula

Heuristic Function	Formula
Manhattan	$\sum \lvert x_i - y_i \rvert$
Octile	$\begin{cases} \sqrt{2} \times dx + dy, & \text{if } dx < dy \\ \sqrt{2} \times dy + dx, & \text{otherwise} \end{cases}$
Cosine	$\sum x_i y_i / \sqrt{x_i^2} \sqrt{y_i^2}$
Diagonal	$(dx + dy) + (1 - 2\sqrt{2} \times min(dx, dy))$
Customized	$\alpha \times \max(dx, dy) +$ $(\alpha - \beta) \times ((\max(dx, dy) + \min(dx, dy)) /$ $(\min(dx, dy) + \gamma + \min(dx, dy))$

x_i and y_i respectively represent x and y positions. dx and dy are the difference between x and y coordinates, respectively. α, β, and γ are parameters. ($\alpha = 1.3, \beta = \sqrt{3},$ and $\gamma = 0.01$)

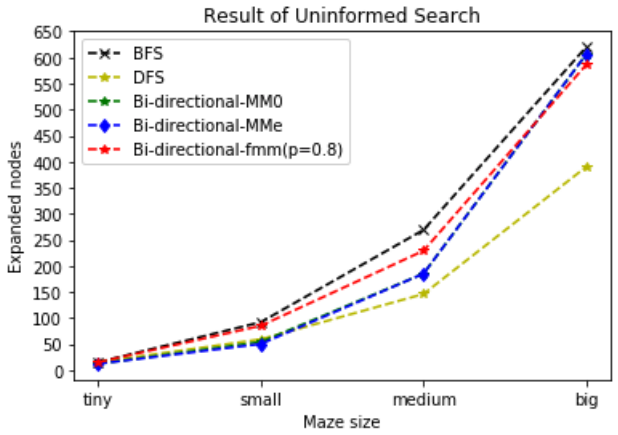

Fig. 1: The Comparison of Uninformed Search Algorithms with Number of Expanded Nodes and Maze Size

III. Results and Discussions

In this section, we show the comparison of the performances among the Uninformed Search (BFS, DFS, MM_0, MM_ϵ, fMM) and Informed Search(A* and MM) with various heuristics. Those representative search algorithms were applied to the Position Search problems in Pacman domain. The first part was that we compared the performance of MM without heuristics and uninformed conventional searches (BFS, DFS). The other part was that we ran A* and MM with heuristics to see how strong/weak heuristics affect their performance with expanded nodes and execution time. The results show the efficiency of each method through relative comparison of heuristics based on the number of expanded nodes and execution time.

A. Comparison in Uninformed Search

Unlike other search algorithms, DFS fails to find an optimal path. Disregarding DFS, we compared BFS and the variants of MM on their performances based on the number of expanded nodes because their total costs were the same as the optimal value. Table II results that MM outperformed conventional uninformed search algorithms. Especially, MM_ϵ found the optimal solution with the least expanded nodes in all the mazes except in big maze. The comparison of uninformed search algorithms with expanded nodes is shown in Fig 1.

B. Comparison in Informed Search

We used various distance-based heuristic functions to see their influence on A* and variants of MM, comparing each algorithm's performance in four size of mazes: the Maze heuristic with A* and MM found the optimal solution to the problem in all the mazes. More detailed information for the comparison is illustrated in Table III.

Manhattan Distance: MM has mostly fewer expanded nodes and better time performance in all mazes. The larger maze size, the more apparent the difference appears between MM and A*, except big maze; only two nodes difference in the small maze, but 37 expanded nodes differed in medium maze. However, in big maze, A* has 57 fewer expanded nodes even though the execution time was slower than MM.

Octile Distance: With Octile Heuristics, there was no significant difference (10) in the number of expanded nodes for smaller mazes. In big maze, however, MM's execution time was faster than A* despite of a slight increase of the number of its expanded nodes.

Cosine Similarity: All the performances of MM in expanded nodes showed better performance than A*. In medium maze, there was 83 difference between A* and MM. MM had also better time performance for all mazes.

Diagonal: MM showed better time performance than A* for three bigger mazes. MM also showed 43 less expanded nodes in medium maze. In big maze, the A* had 28 less expanded nodes.

Customized: Node expansion with A* increased dramatically between the small and medium mazes (a gap of 173 nodes) while MM had much smaller gap (122). And MM took less time.

Maze: Both A* and MM returned the results of the same number of expanded nodes, which is much smaller than by using other heuristics. In other words, both produce optimized

Fig. 2: Comparison of Informed Search Algorithms with Number of Expanded Nodes and Execution Time

TABLE II: The Number of expanded Nodes for Uninformed Search Algorithms

Algorithms					
Maze	BFS	DFS	Bi-directional-MM$_0$	Bi-directional-MM$_\epsilon$	Bi-directional-fMM(p=0.8)
TinyMaze	15	15	**12**	**12**	15
SmallMaze	92	59	54	**50**	85
MediumMaze	269	146	**185**	**185**	229
BigMaze	620	390	606	606	**586**

TABLE III: The Number of expanded Nodes and Execution Time(ms) for Informed Search Algorithms with Heuristic Functions

Algorithms												
Heuristic	Manhattan		Octile		Cosine		Diagonal		Customized		Maze	
Maze	A*	MM	A*	MM	A*	MM	A*	MM	A*	MM	A*	MM
TinyMaze	14	12	13	12	15	12	13	12	12	10	**8**	**8**
(Execution time)	0.787	0.755	0.909	0.891	1.211	0.853	0.757	0.851	0.871	0.801	1.764	5.397
SmallMaze	53	39	55	45	91	54	56	46	85	52	**19**	**19**
(Execution time)	2.737	1.889	2.934	2.056	5.040	2.612	3.150	2.032	4.291	2.530	14.569	142.658
MediumMaze	221	184	223	185	268	185	228	185	258	174	**68**	**68**
(Execution time)	15.127	7.011	13.439	10.459	18.310	7.734	16.968	8.405	17.943	9.034	170.621	1,985.989
BigMaze	549	606	554	606	619	606	578	606	579	574	**210**	**210**
(Execution time)	45.386	26.075	46.298	28.668	53.557	29.278	48.858	27.087	48.853	45.118	3,115.825	185,234.764

path finding solutions. However, MM was a sharply distinct in execution time in this maze. The maze heuristic's execution time took much longer than other heuristics, as shown in Fig 2. This heuristic ranked the highest in all mazes and the ranking results based on the number of expanded nodes in each heuristics are shown in Fig 3.

C. Discussion

As shown in Table II, in Uninformed Search comparison, MM$_\epsilon$ showed the best performance among the bi-directional search algorithms in three smaller mazes. The fMM with p value of 0.8 beats others in big maze. That is, when forward and backward searches meet at closer to the goal in the maze, the search could be more optimal. However, fMM has a disadvantage; it is difficult to find an appropriate value of the parameter p. If p is not set to a proper value in a maze, forward and backward searches are unable to work properly. In other words, there might be some cases that fMM couldn't find the optimal path.

For experiments with Informed Search, MM had overall better performance than A*. Most of the results showed that the number of expanded nodes with MM was smaller not to mention execution time. For Customized distance, MM had the less number of expanded nodes compared to A* in general. The results showed that MM had additional gain that contributed to be better search algorithm in this domain [6]. In Maze heuristics, the number of nodes from A* and MM were the same, which means those heuristics did not provide biased benefit between these two search algorithms.

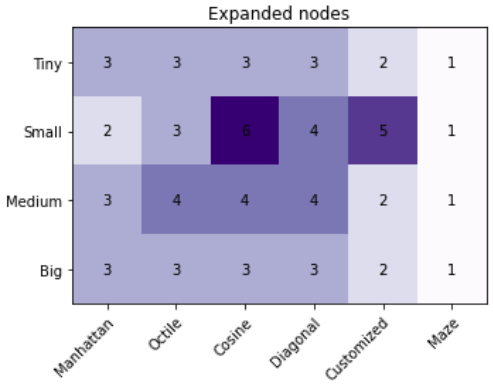

Fig. 3: Ranking of the number of expanded nodes by bidirectional search with heuristics

Fig 3 shows that the performance of MM is ranked so that how each heuristic affected its behavior. As illustrated, by using Maze distance, the number of expanded nodes is the smallest and it ranked as number one in all domains. That is, with Maze heuristic, MM performed with the least number of expanded nodes in all domains. However, despite that advantage of the strongest heuristic, MM had poorer performance than A* in execution time as a trade-off.

In conclusion, the results showed that the performance depends on heuristic functions so that one of heuristic functions can be chosen by considering properties of a domain and a purpose of an implementation.

IV. Conclusions

We implemented MM, MM$_0$, Fractional MM and MM$_\epsilon$. We compared the performance of uninformed search methods including BFS and DFS with different sized mazes. MM$_\epsilon$ showed the best performance among the non-heuristic search algorithms, providing the smallest number of expanded nodes. In informed search algorithms, various heuristic functions were analyzed and performed in Pacman domain. Mainly, MM and A* were representatively shown in the paper, and the results show MM outperformed A* in a majority of times. Performance of a heuristic search algorithm is sensitive to complexity of a domain and heuristic function. For deeper analysis, it is suggested to perform more experimental comparisons on various levels of domains (size level) and heuristic functions (stronger level).

REFERENCES

[1] Barker, Joseph K., and Richard E. Korf. "Limitations of front-to-end bidirectional heuristic search." Twenty-Ninth AAAI Conference on Artificial Intelligence. 2015.

[2] Holte, R. C., Felner, A., Sharon, G., Sturtevant, N. R., and Chen, J. "MM: A bidirectional search algorithm that is guaranteed to meet in the middle." Artificial Intelligence, 2017.

[3] Shaham, E., Felner, A., Chen, J., Sturtevant, and Nathan R. "The Minimaml Set of States that Must Be Expanded in a Front-to End Bidirectional Search." Tenth Annual Symposium on Combinatorial Search. 2017.

[4] Sharon, G., Holte, R. C., Felner, A., and Sturtevant, N. R. "An improved priority function for bidirectional heuristic search." Ninth Annual Symposium on Combinatorial Search. 2016.

[5] Felner, A., Zahavi, U., Holte, R., Schaeffer, J., Sturtevant, N., and Zhang, Z. "Inconsistent heuristics in theory and practice." Artificial Intelligence. 2011.

[6] Sturtevant, N. R., and Felner, A. "A brief history and recent achievements in bidirectional search." Thirty-Second AAAI Conference on Artificial Intelligence. 2018.