Bidirectional Search in Pac-Man Domain

Chi-Yao Huang, De-Ru Tsai, Syed Asad Husain, and Krithish Goli

***Abstract*—This project aims to implement the bidirectional heuristic search algorithm guaranteed to meet in the middle (MM) and compare its properties with others (DFS, BFS, UCS A* search, and bidirectional brute-force search) in the Pac-Man domain. By designing the priority function properly, MM expands the nodes from the $start$ state and the $goal$ state bidirectionally and connects the forward and backward paths in the middle. We provide the technical approach and apply MM to search for the optimal path to the target. The result matches the description of MM. We make multiple experiments to validate the algorithm and provide our viewpoints on the bidirectional heuristic search. In addition, we also try to solve the corner problem via MM. You can see our research results on https://github.com/chiyaohuang/bidirectional-astar**

I. Introduction

As its name implies, a bidirectional heuristic search (Bi-HS) algorithm expands nodes from both the start state and the goal state bidirectionally. Forward and backward searches are conducted simultaneously till their paths meet with each other. Although this search strategy seems effective at first glance, the former research [1] points out that any front-to-end Bi-HS algorithm suffers from limitations that either A* search or bidirectional brute-force search (Bi-BS or MM_0 in this project) may dominate it.

To discuss this in more detail, we should compare the number of nodes expanded by Bi-HS with A* search and Bi-BS. That research demonstrates that Bi-HS expands more nodes than A* search if more than half of the nodes' cost is smaller than the overall optimal cost over two in A* search. Besides, Bi-HS can't outperform Bi-BS if fewer than half of the nodes' cost is smaller than the overall optimal cost over two in A* search. These properties are built on the assumption that Bi-HS meets in the middle if never expanding nodes with a cost that is higher than the overall optimal cost over two. However, here comes the problem. How do we design the Bi-HS guaranteed to meet in the middle? Thus, the successors build MM (a kind of Bi-HS) [2] to address this issue.

In this project, we walk through MM by providing the technical approach, implementing it in the classical Pac-Man domain, and discussing the experiment results.

For Section II, we restate the algorithm and the properties of MM. Then we illustrate the detail of the cost function, the heuristic function, and the priority function for the path-searching problem in the Pac-Man domain. For Section III, we comprehensively compare MM with other search algorithms (DFS, BFS, UCS A* search, and MM_0) respectively by conducting a Student's T-test and showing the difference with some maps. Then we provide our observations and explanations to demonstrate the properties of MM. For Section IV, we make the conclusion eventually.

Algorithm 1 MM in Pac-Man

$g_F(start) := 0$; $g_B(goal) := 0$; $U := \infty$
$Open_F$.push($start$); $Open_B$.push($goal$)
while $(Open_F \neq empty)\ and\ (Open_B \neq empty)$
  $Open_F$.top(); $Open_B$.top()
  $C := min(prmin_F, prmin_B)$
  choose minimum f_F in $Open_F$ as $fmin_F$
  choose minimum g_F in $Open_F$ as $gmin_F$
  choose minimum f_B in $Open_B$ as $fmin_B$
  choose minimum g_B in $Open_B$ as $gmin_B$
  if $U \leq max(C, fmin_F, fmin_B, gmin_F + gmin_B + 1)$
    while $current \neq start$
      $Action_F$.append($action_{current}$)
      $current = Parent_F(current)$
    while $current \neq goal$
      $action$.reverse()
      $Action_B$.append($action$)
      $current = Parent_B(current)$
    return $Action_F + Action_B$
  if $C == prmin_F$ (Expand nodes forward)
    choose $node$ from $Open_F$ with $prmin_F$ and $gmin_F$
    move $node$ from $Open_F$ to $Closed_F$
    for $Child_F$ in children of $node$
      $g_F(Child_F) = g_F(node) + 1$
      if $Child_F \in Open_F \cup Closed_F$
        continue
      $Open_F$.append($Child_F$)
      if $Child_F \in Open_B$
        $U := min(U, g_F(c) + g_B(c))$
  else
    Expand nodes backward

II. Technical Approach

Actually, we can view MM as two A* search agents that expand nodes from both the start state and the goal state. Thanks to this property, we can duplicate the components used in one way and borrow the cost function along with the heuristic function from the former project. For the forward direction, we define the open set (nodes waiting for expansion), the closed set (expanded nodes), the parent node, the child node, and the action leading to the child node as $Open_F$, $Closed_F$, $Parent_F$, $Child_F$ and $Action_F$ respectively. Similarly, we define those as $Open_B$, $Closed_B$, $Parent_B$, $Child_B$ and $Action_B$ in the backward direction.

To simplify the denotation, we only introduce the related equations in the forward direction since the backward direction can be deduced analogously. The cost function $g_F(n)$ is given as the path from the start node to n, where the cost from one node to a neighbor node is value 1. The heuristic function

$h_F(n)$ is given as the Manhattan distance between the start node and n. Similar to A* search, we define $f_F(n)$ as,

$$f_F(n) = g_F(n) + h_F(n) \tag{1}$$

In the traditional A* search, the agent selects the node with minimum $f(n)$ to expand. However, MM introduces a new function named the priority function which is given by,

$$pr_F(n) = max(f_F(n), 2g_F(n)) \tag{2}$$

This equation compares $g_F(n) + h_F(n)$ with $g_F(n) + g_F(n)$ actually. If the cost value is higher than the heuristic value, the priority function returns $2g_F(n)$ instead of $f_F(n)$. This design corresponds to the assumption that Bi-HS meets in the middle if never expanding nodes with a cost that is higher than the overall optimal cost over two. The nodes in the priority queue with $2g_F(n)$ represent the nodes that are beyond the middle range, so the agent will expand them with lower priority. Similarly, we denote these functions in the backward direction as $g_B(n)$, $h_B(n)$, $f_B(n)$, and $pr_B(n)$.

The next step is to decide which direction should be explored at each iteration. Thus, we compare the minimum values $prmin_F$ and $prmin_B$ of each priority queue and take the one with a smaller value to expand. The cost function of MM is given by,

$$C = min(prmin_F, prmin_B) \tag{3}$$

MM's purpose is to find the optimal path with the least cost. The optimal path can be represented by,

$$C^* = d(start, goal) \tag{4}$$

$d(u, v)$ is the actual distance from state u to state v. Thus, the optimal path C^* represents the least cost path from $start$ to $goal$. This optimal solution leads to the following deduction.

During iteration, we need a parameter to record the least cost that MM expands by far so update the utility U as,

$$U := min(U, g_F(c) + g_B(c)) \tag{5}$$

This equation denotes the sum of costs from the $start$ and $goal$ to the current child. U is initialized to be ∞ and updates whenever MM finds a lower cost solution. Then the search process is terminated when,

$$U \leq max(C, fmin_F, fmin_B, gmin_F + gmin_B + \epsilon) \tag{6}$$

where ϵ is the least cost in state space. We can simply define it as value 1 in the Pac-Man domain since every move only takes cost 1. When U is equal to or less than any one of C, $fmin_F$, $fmin_B$, and $gmin_F + gmin_B + \epsilon$, MM should stop searching since the two directional search agents have met in the middle. This condition helps MM to be optimal and can safely stop. During searching, the cost of expanded nodes is equal or less than the optimal cost ($C \leq C^*$), so MM can safely stop when $U \leq C$. Meanwhile, $U \leq C$ implies $C \leq C^*$

In sum, MM satisfies the following properties [2]:

(P1) MM's forward (backward) search never expands a state far ($C^*/2 \leq d(start, s) \leq C^*$) or remote ($d(start, s) \geq C^*$) from $start(goal)$, i.e. its forward and backward searches meet in the middle.

(P2) MM never expands a node whose f-value exceeds C^*.

(P3) MM returns C^*.

(P4) If a path exists from start to goal and MM's heuristics are consistent, MM never expands a state twice.

In the Pac-Man domain, our search agent is Pac-Man itself, and the goal is to search for the optimal path to the food. We can imagine MM as two Pac-Man who want to reunite in the middle between them. By conducting A* search, the optimal forward and backward actions are stored respectively, which means we can obtain the optimal path by combination. Notice that the actions taken in the backward direction should be reversed, as well as the order of the backward action list. Algorithm 1 provides the detail of the implementation.

III. Results, Analysis, and Discussions

A. Results

We create 80 maps for the Student's T-test. There are 4 sizes: tiny (9×9), small (9×18), medium(18×27), and big (36×36), and each size has 20 maps with randomized obstacles. To alleviate the variance of the distance between Pac-Man and the food, we always set Pac-Man at the top right corner and the food at the bottom left corner. Four algorithms (DFS, BFS, A* search, MM_0, and MM) are compared.

The expanded nodes result is shown in TABLE I. MM shows a worse performance compared to DFS, a better performance compared to BFS, and a similar performance compared to A* search if we only care about the average value. We can also compare the similarity between the two algorithms through the T-score. The absolute value of the T-score tells us how different the two groups of samples are, and the larger value indicates that the groups are different. It shows that MM is close to A* search and differs from MM_0, which may surprise people at first glance since MM and MM_0 share the same architecture with the only difference of the heuristic value. However, this result is reasonable if we view these algorithms from the information viewpoint. Since both MM and A* search are informed search algorithms that know where the $goal$ is in advance, they should share similar behavior naturally. As the size grows, the difference between informed and uninformed searches increases because the maps become more complicated.

The score result is shown in TABLE II. MM shows its optimality, which DFS can't provide. The minor difference may be due to a few corner cases that make MM move one more step. These two results demonstrate that MM can expand fewer nodes but keep the same optimality. We will discuss more in the following subsection.

B. Analysis and Discussions

Because we randomly generate the test maps, our t-test report cannot show significant results. Therefore, we pick some meaningful maps to explain our studies deeper.

1) MM vs. DFS: DFS expands the deepest node first and will keep going until reaching a dead end. Since the Pac-Man domain is relatively simple and the maps we provided in the Student's T-test are not that complicated, DFS's average expanded nodes are less than MM's. However, DFS's fatal

TABLE I
EXPANDED NODES COMPARISON BY THE T-TEST

Tiny (MM mean expanded nodes: 19.75)			
	Mean	T-score	P-value
DFS	18.80	0.71	0.49
BFS	26.20	-5.78	0.00
A* Search	19.80	-0.06	0.95
MM_0	20.75	-2.51	0.02
Small (MM mean expanded nodes: 53.50)			
	Mean	T-score	P-value
DFS	39.10	6.13	0.00
BFS	62.95	-4.47	0.00
A* Search	50.85	1.47	0.16
MM_0	56.35	-2.80	0.01
Medium (MM mean expanded nodes: 158.55)			
	Mean	T-score	P-value
DFS	117.00	3.19	0.00
BFS	195.35	-3.79	0.00
A* Search	155.20	0.46	0.65
MM_0	168.65	-2.46	0.02
Big (MM mean expanded nodes: 311.85)			
	Mean	T-score	P-value
DFS	288.30	0.56	0.58
BFS	512.55	-6.56	0.00
A* Search	287.55	1.43	0.17
MM_0	389.45	-5.42	0.00

TABLE II
SCORE COMPARISON BY THE T-TEST

Tiny (MM mean score: 497.5)			
	Mean	T-score	P-value
DFS	495	4.46	0.00
BFS	497.5	NaN	NaN
A* Search	497.5	NaN	NaN
MM_0	497.5	NaN	NaN
Small (MM mean score: 484.6)			
	Mean	T-score	P-value
DFS	480.8	5.47	0.00
BFS	484.8	-1.45	0.16
A* Search	484.8	-1.45	0.16
MM_0	484.8	-1.45	0.16
Medium (MM mean score: 459.4)			
	Mean	T-score	P-value
DFS	443.7	8.34	0.00
BFS	459.5	-1	0.33
A* Search	459.5	-1	0.33
MM_0	459.5	-1	0.33
Big (MM mean score: 432.4)			
	Mean	T-score	P-value
DFS	380.6	9.44	0.58
BFS	432.4	NaN	NaN
A* Search	432.4	NaN	NaN
MM_0	432.4	NaN	NaN

defect is the lack of optimality. Besides, DFS expands more nodes than MM when many branches lead to dead ends. We provide an extreme case for DFS that initially selects the less effective direction in Fig. 1. MM outperforms DFS in this map no matter whether we compare them by the time complexity (185:313 expanded nodes) or the optimality (461:197 scores).

2) MM vs. BFS and UCS: Since BFS and UCS share similar behavior in the Pac-Man domain due to the cost value 1 between two neighbors, we can discuss them together. These non-directional search algorithms tend to expand more nodes than MM, but find the optimal solution like MM. We can

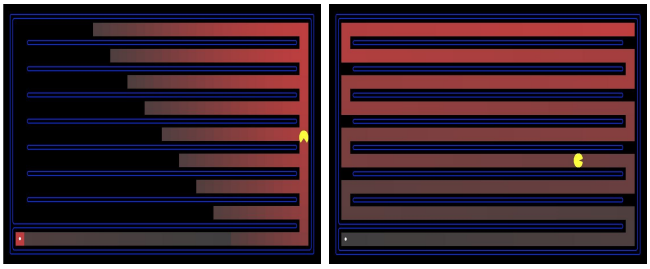

Fig. 1. Compare MM (left) with DFS (right) by expanded nodes and scores in an extreme case that MM outperforms DFS.

observe these properties in most maps. Fig. 2 provides a common case that BFS (or UCS) gradually expands most nodes (313) from the $start$ to the $goal$. On the contrary, MM explores bidirectionally to avoid the useless region and expands fewer nodes (188). Both of them get the same score 403.

3) MM vs. MM_0 and A search:* MM_0 is a kind of Bi-BS that shares the same design with MM but doesn't consider heuristic values. The red gradient in the Pac-Man domain represents the order of nodes in the priority queue. The deeper the color is, the earlier the node is expanded. We can observe that the color decays from the $start$ and the $goal$ bidirectionally in MM_0, which means MM_0 expands nodes from both sides by turn. Since MM_0 doesn't consider the heuristic function (all heuristic values are 0), the nodes closer to the $start$ or the $goal$ are at the top of the priority queue. On the other hand, MM considers the heuristic function. Since our MM expands the $start$ during the first iteration, $start$'s successors may be at the top of the priority queue compared to the $goal$'s once the agent moves close to the middle. We can observe this phenomenon in some maps that the color closer to the $goal$ is lighter than the color closer to the $start$.

A* search expands more nodes than MM in circumstances where nodes share a similar f value according to the distance from the $start$, but only a few can lead to the $goal$. We can observe this property in Fig. 3 where MM expands fewer nodes (169 vs. 305) but gain the same score (461). This observation implies that MM has higher effectiveness if the states closer to the $goal$ are simpler than those closer to the $start$.

In addition, [1] makes two conclusions for bi-directional search:

(BK1) A* search will expand fewer nodes than Bi-HS if more than half of the nodes expanded by A* search have $g \leq C^*/2$, where C^* is the optimal solution cost.

(BK2) If fewer than half of the nodes expanded by A* search using heuristic h have $g \leq C^*/2$, then adding h to Bi-BS (bi-directional brute force search) will not decrease the number of nodes it expands.

For **BK1**, we use an example to reproduce the relation between the expanded nodes and the path cost g. We set $start$ and $goal$ to the top right corner and the left down corner in Fig. 4. The height and width of the layout are 18 and 36. We use the Manhattan distance as our heuristic cost. The optimal solution C^* is (18 - 2) + (36 - 2) - 2 = 48 (subtract boundaries and the start and the goal). The path cost g of each node is

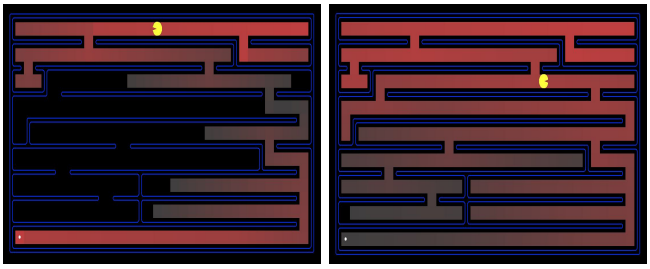

Fig. 2. Compare MM (left) with BFS (right) by expanded nodes and scores in a common case that MM expands fewer nodes and has the same optimality.

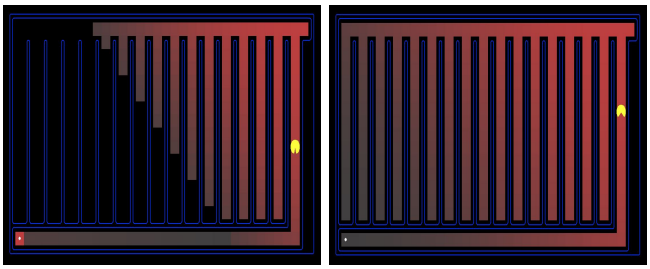

Fig. 3. Compare MM (left) with A* search (right) by expanded nodes and scores in the case that MM expands fewer nodes and has the same optimality.

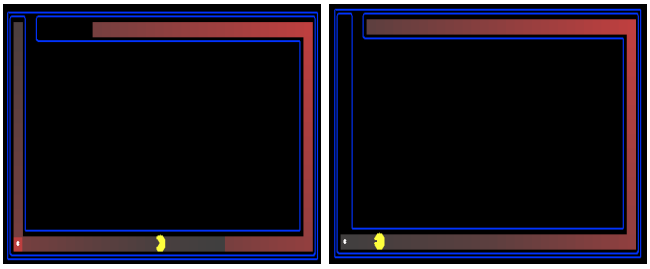

Fig. 4. **BK1**: MM (left) expands more nodes than A* search (right) in this case.

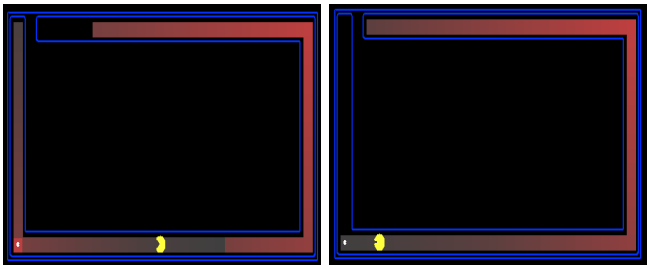

Fig. 5. **BK2**: UFS and MM0 have the same performance as A* and MM under the condition that more than half of the nodes expanded by A* have $g \leq C^*/2$. The heuristic cost will not decrease the number of expanded nodes in this case.

also the Manhattan distance from $start$. Therefore, before the middle point, The g of each node is less than $C^*/2$. The experiment result shows that A* expands only 78 nodes but MM expands 87 nodes. Uni-HS expands fewer nodes than Bi-HS.

For **BK2**, we remove the heuristic cost to make A* and MM become UFS and MM. Fig. 4 shows the experiments are the same as Fig. 5, which means adding the heuristic h will not decrease the number of expanded nodes.

According to **BK1** and **BK2**, we can assume that half of the expanded nodes of Bi-HS must satisfy $g \leq C^*/2$. In other words, both of the forward and backward searches of Bi-HS never travel nodes whose $g > C^*$. Therefore, the forward and backward searches are guaranteed to meet in the middle because they will not expand further nodes.

IV. Corner Problem

We also try to solve the corner problem by MM. In the corner problem, the agent has to find the optimal path that passes through four corners. The bottleneck is: which corner should be taken as the $goal$? We modify the heuristic function for the Pac-Man (n) in the forward direction as,

$$h(n) = d(n, c_1)_{min} + d(c_1, c_2)_{min} + ... + d(c_3, c_4)_{min} \quad (7)$$

and set the heuristic function for one of the corners in the backward direction in the same way. Notice that n represents a set corner for the backward direction, and the Pac-Man's location is taken as one of the corners.

Our first approach is to conduct MM directly by the heuristic function mentioned above. We pick one corner as the $goal$ and see whether the agent can find all corners. Unfortunately, the result shows that the agent can only reach the set corner and then stay there, let alone find the optimal path.

Our second approach is to expand all corners at the same time, which is similar to [3] Thus, we have four backward open sets and four backward closed sets now. The conjecture is that both forward and backward searches should follow the heuristic value to find other corners. If they connect, then we may get the optimal path passing through all corners. Unfortunately, the result shows that the agent can only reach the closest corner and stay there. We assume that MM should be conducted dynamically according to the least heuristic value to four corners. If the agent reaches the first corner, then runs MM again till finding all corners. Due to the limited time, we skip this approach.

V. Conclusions

In this project, we make a detailed review of "Bidirectional Search That Is Guaranteed to Meet in the Middle" [2]. We first provide the key concept of it and implement the algorithm of MM and MM0. Its algorithm with a clearer pseudo code is restated. Second, we offer comprehensive experiments in the Pac-Man domain and analyze the results with meaningful discussion. All materials can be accessed via our github. In advance, we also provide our rough idea to use MM to solve the corner problem referred to [3].

References

[1] J. Barker and R. Korf, "Limitations of front-to-end bidirectional heuristic search," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29, no. 1, 2015.

[2] R. Holte, A. Felner, G. Sharon, and N. Sturtevant, "Bidirectional search that is guaranteed to meet in the Middle," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, no. 1, 2016.

[3] D. Atzmon, J. Li, A. Felner, E. Nachmani, S. Shperberg, N. Sturtevant, and S. Koenig, "Multi-directional heuristic search," Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, 2020.