New algorithm for digital way-finding map

IOP Conf. Series: Materials Science and Engineering **407** (2018) 012074 doi:10.1088/1757-899X/407/1/012074

New algorithm for digital way-finding map

M Aria

Electrical Engineering Department, Faculty of Engineering and Computer Science, Universitas Komputer Indonesia, Jalan Dipati Ukur No 112 – 116 Bandung, Indonesia

muhammad.aria@email.unikom.ac.id

Abstract. The purpose of this research was to develop a new algorithm to find the shortest pathfinding that can be implemented on a digital way-finding map. The standard algorithm used for pathfinding is Dijkstra and A*. But for a large search space, the A* and Dijkstra need a large amount of CPU and memory resources. Whereas pathfinding algorithm is usually embedded in devices with limited memory. So, we make the algorithm which requires less memory. In order to make a less memory algorithm, we modify the Breadth First Search (BFS) and Bidirectional Search (BS) algorithm and use the cost graph and modify adjacency matrix. We have tried to reduce the runtime of our algorithm using the modify adjacenvy matrix. The proposed algorithm is then tested for path finding at the institution that has several buildings with many floors where staircase position is not uniform on each floor. The test results were then compared with Djikstra, A*, IDA*, and HPA*. The experiment results showed that our algorithm is faster than the A*, IDA* and Djikstra algorithm. This is because the algorithm that we designed only need to make modify adjacenvy matrix once, so it can reduce the computation time. However, for cases involving multiple floors, our algorithm is not faster than HPA *, but approached. From the test results can be concluded that our algorithm can be implemented in the case of searching the route on buildings that have many floors.

1. Introduction

Digital wayfinding maps are interactive touch screen monitor used in buildings or campuses to provide directional information to user or visitors [1]. Way-finding or path-finding algorithm describes a process to finding the optimal path from some location to destination location, which means to find the shortest, the simplest or the cheapest path. Digital wayfinding system usually embedded in devices with limited memory. So we need the pathfinding algorithm which need less amount of CPU and memory resources.

Many pathfinding algorithms have been studied for more than a decade. The popular for heuristic-based algorithms are A*[2] and Dijkstra's algorithm [3]. The two algorithms return an optimal path [4]. The A* algorithm is made by Hart and Nilsson [5]. Dijkstra's is simple than A*, but A* algorithm is faster than Dijkstra's algorithm. This is because Dijkstra's implement the Greedy Best First Search algorithm while A* implement the Best First Search algorithm. Dijkstra's disadvantage is there spent a lot of time waste because it was a blind search [6,7]. Zhang at [8] gives the following illustration: when the number of the nodes is 3000, it takes 36 MB of memory, and if the number of the nodes is 6000, it takes 144 MB of memory. From this example it can be seen that the memory requirements will increase exponentially. To speed up the pathfinding process, Adi Botea in 2004 developed Hierarchical Pathfinding A * (HPA *) [9]. HPA* combines clustering algorithms and pathfinding

algorithms. HPA* works by creating abstract graph based on two-dimensional grid. The working principle of HPA* is divides the search problem into some smaller sub-problems, then caching results for each sub-problem [10, 11]. Botea [9] shows that HPA * can be 10 times faster than A * [9]. The disadvantages of HPA*are the cost increases significantly when adding a new abstraction layer. The other researchers saw that A* could not solve a large problem because it was running out of memory. To reduce space requirements in A*, Iterative Deepening A* (IDA*) developed by Korv in 1985 [12]. IDA* can obtain optimal solutions such a A* but require linear memory usage in the cost of the solution. IDA* implement the idea of iterative deepening search algorithm. The advantages of IDA* is require less memory than A*. But the disadvantages of IDA* is wasted search.

This paper presents a new algorithm to make the pathfinding process faster and to reduce the memory requirements simultaneously. In our previous study [13], we have investigated the minimal hardware and software requirements for the implementation of the Digital Interactive Wayfinding Map. In this research, we modify the Breadth First Search (BFS) and Bidirectional Search (BS) algorithm to make a fast and less memory algorithm, to reduce the runtime of the proposer algorithm, we use the cost graph and modify adjacency matrix. Our algorithm is then tested for searching optimal path at the institution that has several buildings with many floors where staircase position is not uniform on each floor. We compare the test results with Djikstra, A*, HPA*, and IDA*.

2. Method

The algorithm starts with start node and search in all four directions of node (up, down, left and right). This process continues until destination node is reached. In the process, cost of each node which is equivalent to the index value to achieve that node is stored. The shortest path is searched by backtracking from the destination node back to the initial node through nodes that have a minimum value compared to its neighbour nodes. There are several resemblances between the algorithm developed in this study and the algorithm presented by Ahlam [14] , Geroge [15], and Nawaf [16], although there are some important modifications.

To demonstrate the principle of the propose algorithm, we will use the illustration of its application in cases such as the graph in figure 1. In the picture there are one start node (S) and four destination nodes (A-D). The program will add 8 intersection nodes (F - L) as in figure 2. The steps sequence of the propose algorithm is as follows.

- Creating an cost graph containing the index value from the starting node to each other node until all node is filled. Figure 3 show example of a cost graph.

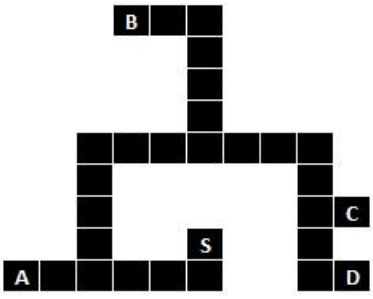

Figure 1. Simple graph to explain the principles proposed algorithm.

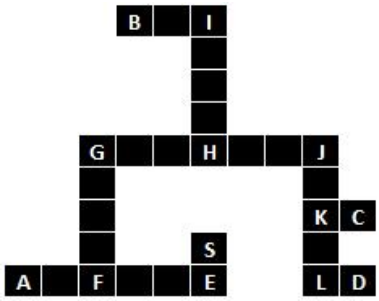

Figure 2. Addition intersection node on the graph.

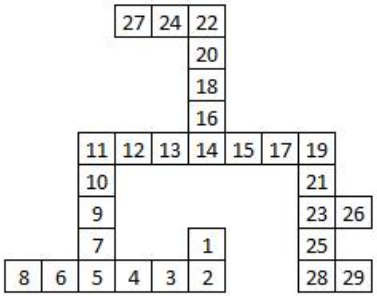

Figure 3. Making a cost graph.

- Making the “modify adjacency matrix” between nodes in the cost graph that has been created as shown in figure 4. This matrix informs which nodes are interconnected and from those nodes, which nodes are closer to the starting node.

- For example, to find a route to node B from node S, it is done from column / row B. In the column / row B, we will look for the smallest value of the cell. In the row / column B we can be seen the smallest value is 27 on a row / column I. This means that from node B must go to node I. Furthermore search the smallest value in the row / column I who turns 22 on row / column H . This Means that from node I must go to the node H. By continuing this process, it will get the path that must be taken is B - I - H - G - F - E - S. To find a route from S to B, then the route from B to S can be reversed order.

Due to the implementation of this algorithm is the "You Are Here Map", then the starting node will always be the same. So that the “The modify adjacency matrix between node” quite made one time only. To find a route from node S to other nodes quite repeat step 3 only. That's why our algorithm can work faster than any other algorithm for this wayfinding case. We modified the above algorithm using the bi-directional Search principle if the start node and destination node are located on different floors (See Figure 4).

	S	A	B	C	D	E	F	G	H	I	J	K	L
S	1					2							
A							8						
B										27			
C												26	
D													29
E	2						5						
F		8				5		11					
G							11		14				
H								14		22	19		
I			27						22				
J									19			23	
K				26							23		28
L					29							28	

Figure 4. The modify adjacency matrix between nodes.

3. Results and discussion

To verify our proposed algorithm, we performed two steps test. First, we compared the path length from our algorithm with Dijkstra’s, A*, HPA* and IDA* algorithm; second, we measure the computational cost from five algorithms. The proposed algorithm is then tested for path finding at the institution that has several buildings with many floors where staircase position is not uniform on each floor. The algorithm was tested on the implementation of the wayfinding system in the building (You Are Here Map). The characteristics of the “You Are Here Map” is have a single starting node (You Are Here node) and has many target node (the location of each room in the building).

3.1. Comparison the path length

Firstly, our proposed algorithm we compared with the other algorithm to test its ability to produce the best route. We tested it in 10 cases shown in table 1. From the table 1, we can see that our proposed algorithm can produce the path as well as another algorithm. Because our algorithm uses the principle of BFS, the completeness and optimality characteristics are also owned by our proposed algorithm.

3.2. Comparison in *time*

In the table 2 we present the results of average execution time from the five tested algorithms. Testing is done in the case of "You Are Here Map", with the amount of testing is 200 times. From the experimental results presented in Table 2, we can see that HPA* is the fastest algorithm and almost two times faster than the Dijkstra algorithm. However, our algorithm is not faster than HPA*, but approached. And our algorithm can be faster than Dijkstra, A* and IDA* algorithm. These results are

similar to those obtained by Noori [17]. He also reported that the execution time of IDA* is the longest and the fastest is HPA*.

Our algorithm works fast because for every run it does not need to repeat the entire process from step 1, but simply repeat step 3 only. So, our algorithm only need to make modify adjacenvy matrix once and the same matrix can be used repeatedly, so it can reduce the computation time (See Table 1).

Table 1. Comparison in path length from five tested algorithm.

	Comparison in path length				
	Proposed Algorithm	Dijkstra Algorithm	A* Algorithm	HPA* Algorithm	IDA* Algorithm
Case 1	36	36	36	37	36
Case 2	67	67	67	69	67
Case 3	85	85	85	87	85
Case 4	27	27	27	27	27
Case 5	66	66	66	68	66
Case 6	25	25	25	25	25
Case 7	82	82	82	84	82
Case 8	35	35	35	36	35
Case 9	27	27	27	28	27

Table 2. Average execution time with the amount of testing is 200 times in the case of "You Are Here Map"

Algorithm	**Execution time (ms)**
Proposed Algorithm	1.782
Dijkstra	3.849
A*	2.988
HPA*	1.647
IDA*	7.932

This can be compared with the djikstra algorithm that works as follows [2].
Step 1: The distance value to each node is set. The current node is the starting node. The state of all nodes is unvisited.
Step 2: Checked all neighboring nodes of the current node:
The tentative distance (Dt) through the current node is calculated
If Dt is smaller than D (previously recorded distance) then D = Dt
The current node is marked as visited (the shortest distance are found)
Step 3 : The next current node is selected from unvisited node that have the smallest distance.
Step 4 : Algorithm will stop after all nodes are marked as visited. If not, then the algorithm will be repeated from step 2.
The Dijkstra algorithm must reset all memory (make all nodes marked as unvisited) and repeat the entire process from step 1 for each path search.
Likewise with the A * search algorithm that has the following work processes [5].
Step 1 : The first parent node is the starting node
Step 2: Extended n child node from the parent node
Step 2: Calculate the $G(n)$, $H(N)$, and $F(n)$ value for new add child nodes
Step 3 : Checking whether there is a destination node between n new child node. If not, then proceed to step 4, but if the destination node is found, then backtracking from the destination node to the initial node through the node has the smallest $F(n)$ value

Step 4: Selected the smallest $F(n)$ value from all child nodes as the next parent node. Repeat again from Step

The A* search algorithm should start the search process from the starting node, re-create the search tree from the starting node and the value of $F(n)$ for each node must be recalculated for each path search. This causes the A * search algorithm to require a longer search time than our algorithm.

An example of the search results of the algorithm we designed is shown in Figure 5. From figure 5 it can be seen that our algorithm can be implemented in the case of route searching on buildings that have many floors (See Figure 5).

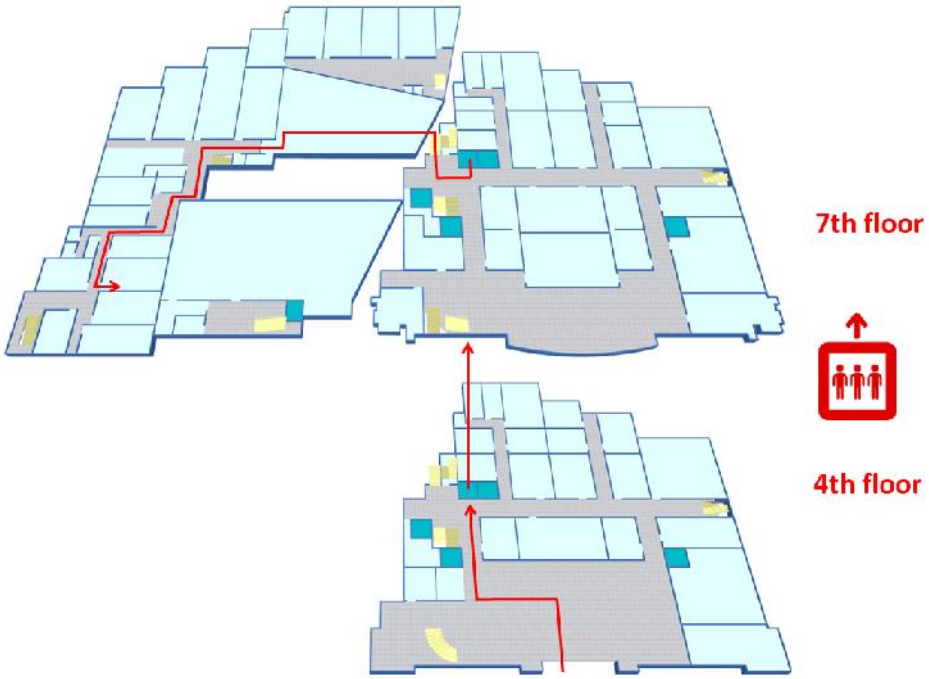

Figure 5. Example implementation of our algorithm for building that have many floor.

4. Conclusion

We have presented an alternative algorithm for path finding problems. Based on test results, we can see the proposed algorithm able to produce an optimal path like the other algorithm. From the second test, we can see the proposed algorithm is faster than Dijkstra, A* and IDA* algorithm. However, our algorithm is not faster than HPA *, but approached. From the test results can be concluded that our algorithm can be implemented in the wayfinding case on buildings that have many floors.

References

[1] Tüzün H, Telli E and Alır A 2016 *Usability testing of a 3D touch screen kiosk system for way-finding* Computers in Human Behavior **61** p 73-79

[2] Ferguson D, Likhachev M and Stentz A 2005 *A* Guide to Heuristic-based Path Planning *Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS)* 1- 10

[3] Dijkstra E 1959 *A Note on Two Problems in Connexion with Graphs* Numerische Mathematik 1 269 – 271

[4] Gelperin D 1977 *On the optimality of A** Artificial Intelligence **8** 1 69–76

[5] Nilsson N J, Hart P E and Raphael B 1968 A Formal Basis for the Heuristic Determination of Minimum Cost Paths *IEEE Transactions on Systems, Science, and Cybernetics* **4** 2 p 100–107

[6] Goyal A, Mogha P, Luthra R and Sangwan N 2014 Path Finding : A* or Dijkstra's?

International Journal in IT and Engineering **2** 1 p 1 – 15
[7] Sreekanth R K 2012 Artificial Intelligence Algorithms *IOSR Journal of Computer Engineering (IOSRJCE)* **6** 3
[8] Zhang F, Qiu A and Li Q 2005 Improve on Dijkstra shortest path algorithm for huge data *ISPRS* **XXXVII-7/C4** p 313-316
[9] Botea A, Muller M and Schaeffer J 2004 Near, Optimal Hierarchical Path-Finding *Journal of Game Development* **1** 1 p 1–30
[10] Jansen M R and Buro M 2007 HPA* Enhancements *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference* p 84–87
[11] Holte R C, Holte M B, Zimmer R M and MacDonald A J 1996 Hierarchical A*: Searching Abstraction Hierarchies Efficiently *AAAI 96 Proceedings of the thirteenth national conference on Artificial intelligence* p 530-535
[12] Korf R 1985 Depth-first iterative deepening: An Optimal Admissible Tree Search* *Artificial Intelligence* **27** 1 p 97-109
[13] Aria M, Mulyana A and Albar D 2015 Smart Interactive Campus Map for Universitas Komputer Indonesia *The 2nd International Conference on Applied Information adn Computer Technology* p II.51 – II.56
[14] Ahlam A, Mohd A S, Khatija R and Parvin S 2015 An Optimal Hybrid Approach for Path Finding *International Journal in Foundations of Computer Science & Technology (IJFCST)* **5** 2 p 47–58
[15] George L 2013 Quantitative Comparison of Flood Fill and Modified Flood Fill Algorithms *International Journal of Computer Theory and Engineering* **5** 3 p 503–508
[16] Nawaf H B, Sinan S M A and Mustafa A S N 2016 Pathfinding in Strategy Games and MazeSolving Using A* Search Algorithm *Journal of Computer and Communications* **4** p 15–25
[17] Noori A and Moradi F 2015 Simulation and Comparison of Efficency in Pathfinding algorithms in Games *Ciência e Natura* **37** 2 p 230–238