ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 10, Issue 4, pp: (3-9), Month: October - December 2022, Available at: www.researchpublish.com

Path Optimization within an Interior Space or Building: Application of Shortest-Path Algorithm

Justin Choi

[1,2]Gyeonggi Suwon International School

[1,2] Suwon City, South Korea

DOI: https://doi.org/10.5281/zenodo.7198822

Published Date: 13-October-2022

Abstract: **Due to the rapid development of human industry and the increase in human population, high-rise buildings having complex structures appear to be expected. To maintain the sanitation of such facilities, workers make a full route to check a countless number of trash cans without knowing the emptiness of the trash can, which results in inefficiencies for workers to travel unnecessary trips. Hence, this research paper explores pathfinding algorithms, such as the A* algorithm, RRT algorithm, Floyd algorithm, Dijkstra Algorithm, and Bellman-Ford Algorithm, that can be successfully and efficiently applied to the interior building and function as navigation for workers to travel in the optimal path, assuming the smart trashcan provides the information of its emptiness. This will allow workers to travel on the optimal path, saving labor costs and time and improving overall sanitation and management of trash in high-rise buildings. The evaluation of the pathfinding algorithms has been done on the grid/graph form of the simplified interior building with varying vertices and edges. More profound research on pathfinding algorithms has been presented in this research paper.**

Keywords: **path optimization, pathfinding algorithm, interior space, smart trash can.**

1. INTRODUCTION

Imagine how many trash cans are there in workplaces in high-rise buildings or huge shopping malls. Emptying trash cans in those places in a timely manner will require a significant number of workers. The workers have to make regular trips to each and every trash can to check whether a trash can is full or not, and at each time they should make a full route for the places they are responsible for. This results in inefficiencies for the workers making unnecessary trips of collection or unintentional neglecting of overflowed trash cans. If trash cans are smart so that they can tell their locations and their fullness, then handheld devices such as smartphones can show the map of the route of trash cans that the workers should visit to empty. The suggested route should be optimal in total travel time and distance for the workers or for automated collecting machines. Before developing a fully operating smart connected system, the optimal path calculation method should be either developed or existing methods should be evaluated.

2. OBJECTIVES AND AIMS

The expected benefits of this smart connected system are, but not limited to, 1) improve working conditions for the existing maintenance staff by increasing efficiency, 2) produce tangible advantages in terms of enhanced sanitation, and 3) reduce the running cost of human resources by reducing the number of total maintenance staff.

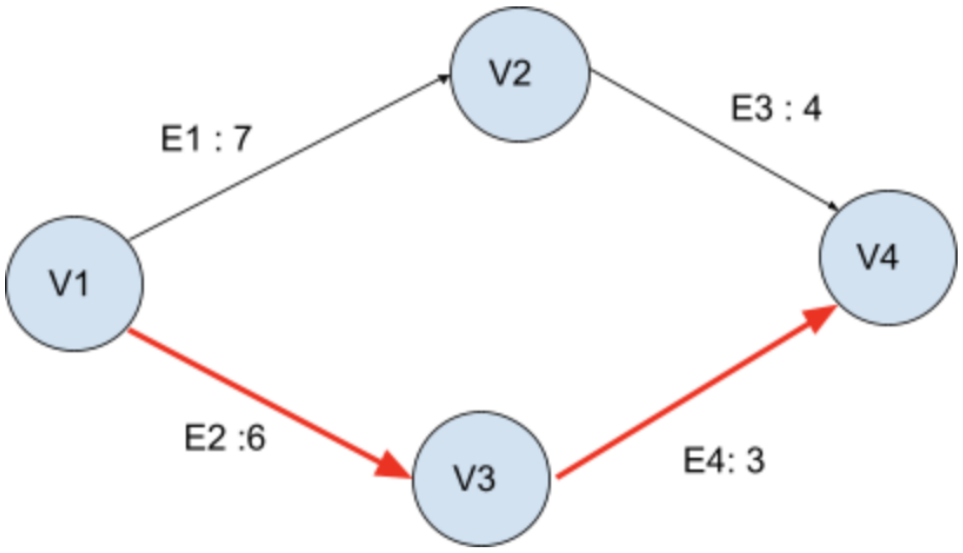
The grand goal of the project is to build a smart system that connects all trash cans (ideally, all means all trash cans in the world ultimately), which is comprised of three main components: 1) smart trash cans that can sense the status of fullness and transmit its status including location and other related information to the internet server system, 2) a smartphone application that will suggest to a worker or to an automated garbage collecting system the route of the path, and 3) a fast path optimization algorithm that will run in the server system.

Among the whole scope of the work towards building the fully operating system, this study aims to 1) explore the feasibility of the five path optimization algorithms that are already developed by others, and 2) isolate the most efficient path optimization algorithm in terms of time and distance. For the evaluation of algorithms, a small scaled real world case is used.

3. BACKGROUND AND SIGNIFICANCE

The shortest path problem is a task for finding the shortest path from the starting point to the destination. To determine the shortest path, algorithms use various mathematical approaches.

One of the most popular mathematical approaches to find the shortest path is graph theory. In graph theory, there are three essential components: vertices, edges, and distances/weight of edges. In the mathematical expression, vertices and edges in graph G are expressed as G=(V, E). Vertices are positions or points; in other words, they are full trash cans that cleaners must check. Edges are the lines connecting vertices; in other words, they are the routes from full trash cans to other full trash cans that cleaners travel. Distances/weight of edges are distances between the trash cans. If the graph contains the distance between the vertices expressed as D(v1,v2), then the shortest path will be easily found. For example, in graph G, vertices are V = {v1, v2, v3, v4} edges are E = { e1, e2, e3, e4}. e1= (v1, v2), e2= (v1,v3), e3=(v2,v4), e4=(v3,v4), and distances are D1 (v1, v2) = 7 ,D2 (v1,v3) = 6, D3(v2,v4) = 4, D4(v3,v4) = 3 [5].



[Figure 1] Example Mechanism of Graph Theory

Then, Let the starting vertice be v1, and the ending vertice be v4. Then, by comparing the distances of every path, the shortest path that has the least distance will be determined, which is v1-> v3-> v4. The process of determining the shortest path was simply adding distances while traveling vertices; hence, the shortest path had a distance of 9, whereas another path had a distance of 11 [5].

On the other hand, some algorithms do not use graph theory; rather, they use simpler concepts: coordinates in two-dimensional planes to determine the shortest path. Compared to algorithms using graph theory that require vertices, edges, and distance, they only require coordinates of starting point and ending point in the form of (x,y). They rapidly explore coordinates that have the least distance from the starting point to the ending until they reach the ending point. Hence, by calculating the total distance from starting point to the ending point, algorithms are able to show the shortest path. Although there are many mathematical approaches to calculate the shortest path between two points, one of the most well-known methods is the Pythagorean theorem. For example, if there are no obstacles, and the starting point and ending point are given as (0,0) and (5,5), then the optimal path can be calculated by the Pythagorean theorem [5].

However, there are always obstacles in the buildings, such as doors and walls. Since algorithms using graph theory do not work with coordinates in a two-dimensional plane, but with vertices and edges, information about obstacles must be given. There are two ways to solve this problem. The first is to input the distance between the trash can considering the obstacles. The second is to put extra vertices on the corner so that lines connecting vertices can be straight. I will apply the second solution because the derived shortest path should be feasible in buildings of different structures. On the other hand, for algorithms using coordinates, it is able to input a whole two-dimensional plane of the building, including obstacles [5].

In short, there are clear pros and cons between the algorithms using different mathematical approaches. The algorithms using graph theory show the shortest path quickly and are capable of having more than one vertice in one input, but obstacles must be considered carefully. On the contrary, the algorithms not using graph theory find the path considering the obstacles, but they take more time than the algorithms using graph theory. Also, they can not accept several ending points or destinations, which means that algorithms have to be executed several times until they check all the trash cans. Additionally, the path found is not always guaranteed as the shortest path because while the algorithms using graph theory connect trash cans with already input distances, the algorithms not using graph theory need to calculate the distances, which hugely depends on the methods of calculations that algorithms use [5].

Although there are many algorithms that solve the shortest path problem, only several algorithms will be discussed in this paper: Dijkstra's Algorithm, Bellman-Ford's Algorithm, Floyd's Algorithm, A* Algorithm, and RRT* Algorithm. The representative algorithms using graph theory are Dijkstra's Algorithm, Bellman-Ford's Algorithm, and Floyd's Algorithm, and the representative algorithms not using graph theory are A* Algorithm and RRT Algorithm [3], [5].

[Types of Shortest-Path Algorithms]

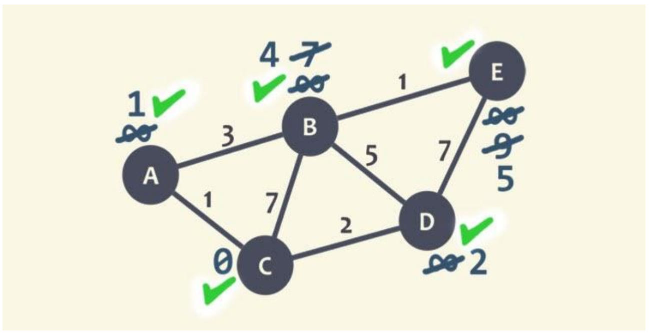
➢ Dijkstra's Algorithm

Dijkstra's Algorithm works by comparing the distance of paths and finding the shortest path among them.

First step: The algorithm initially sets up the starting vertice and the destination. For example, the initial node is 0, and the neighboring nodes have infinity as the minimum distance.

Second step: The algorithm explores the nodes by adding the current node's distance and exploring the node's distance. Then, it compares the current distance with the minimum distance. If the current distance is shorter than the minimum distance, then now the current distance becomes the minimum distance.

Third step: If all the neighboring nodes are explored, then the algorithm chooses the unvisited node to explore. Repeat the second step until there are no unvisited nodes [4][7].



[Figure 2] Mechanism of Dijkstra's Algorithm [6]

Source: Tyagi, Neelam "Marked Node E as Visited." Dijkstra's Algorithm: The Shortest Path Algorithm, 14 Dec. 2020.

➢ Bellman-Ford Algorithm

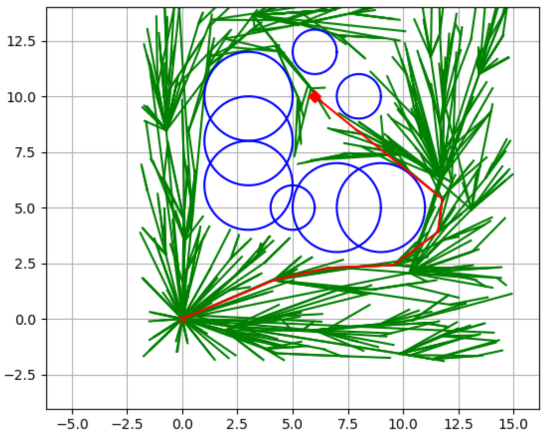
Bellman-Ford's Algorithm is a modified version of Dijkstra's Algorithm that distances can consist of negative values. Also, there is a change: there are additional loops for negative values. Comparably, Bellman-Ford's Algorithm is slower than Dijkstra's [9].

➢ Floyd's Algorithm

Floyd's Algorithm is also a modified version of Dijkstra's Algorithm, but it finds the shortest path between every vertice. Although negative values are available, there is no additional loop for negative values. Since Floyd's algorithm compares the distance between every vertice, its calculation time will hugely vary on the number of vertices [8].

➢ **RRT* Algorithm**

The RRT* algorithm creates points randomly, and based on the points created, it rapidly increases the number of trees until it reaches the destination. After finding the destination, the RRT* algorithm selects the tree that has the lowest cost/distance to the starting point, which works similarly to Dijkstra's Algorithm [1].



[Figure 3] Mechanism of RRT* Algorithm [2]

Source: Bot, Jekyll. Rapidly Exploring Random Tree (RRT), 24 Apr. 2020.

➢ **A* Algorithm**

A* Algorithm finds the shortest path by rapidly exploring grids or coordinates and selecting the nodes that have the least 'f' value. A* repeats this process until it reaches the ending point. Here, 'f' value is equal to the sum of 'g' value and 'h' value

'g' = The movement cost from the starting point to a currently selected point, following the path generated from the starting point.

If the current point is 2 grids apart from the starting point, then the g value is 2.

'h' = Estimated distance from starting point to the destination.

If the starting point is (0, 0), and the destination is (5, 3), then the H value is 34, 52 + 32= 34. However, most of the time, root in H value is ignored because whether root is present or not, it does not affect the result.

Therefore, the algorithm consistently calculates 'g' value and selects the node that has the least 'f' value until it reaches the destination [10].

4. RESEARCH DESIGN AND METHODS

Qualitative analysis based on data acquired from the mathematical simulation will be the main analytical method of this research. Preceding such simulations, however, this study will first involve designing three-floor plans of varying complexity. The floor plans will be adapted from existing buildings, although some simplifications may be introduced for experimental purposes.

Within the floor plans, three types of reference points will be specified: point of departure, point of exit, and targets (full trash-cans). The five aforementioned algorithms will be tested and evaluated upon the following criteria: 1) total distance of shortest-distance path and 2) calculation time.

For Dijkstra's algorithm, Bellman-Ford algorithm, and Floyd algorithm, the calculation time will be the main evaluation since the same inputs will be given. However, for A* algorithm and RRT* algorithm, the total distance of shortest-distance path will be the main evaluation as their mathematical approaches are different from those of other algorithms.

Raw data collected will be tabulated in a frequency table consisting of estimated time, distance, and building structure, as a means to analyze the effectiveness of each simulated algorithm.

5. PRELIMINARY RESULTS

Vertices: Number of Trash Can

Edges: The connection between the Trash Can

Weight of Edges/Distance: Distance between Trash Can

To evaluate the effectiveness of algorithms using graph theory for the shortest path, three different types of floor plans were used. For the first floor plan, the building is constructed of a total of three floors, and on each floor, there are ten trash cans. Also, the paths available for the cleaners are fifty, which means that there are fifty ways for the cleaner to pick up trash.

For the second floor plan, the number of floors remains the same, but the paths available are one hundred.

For the third floor plan, the number of floors is ten floors, which means there are one hundred trash cans in one building, and the paths connecting the trash cans are one-hundred-fifty.

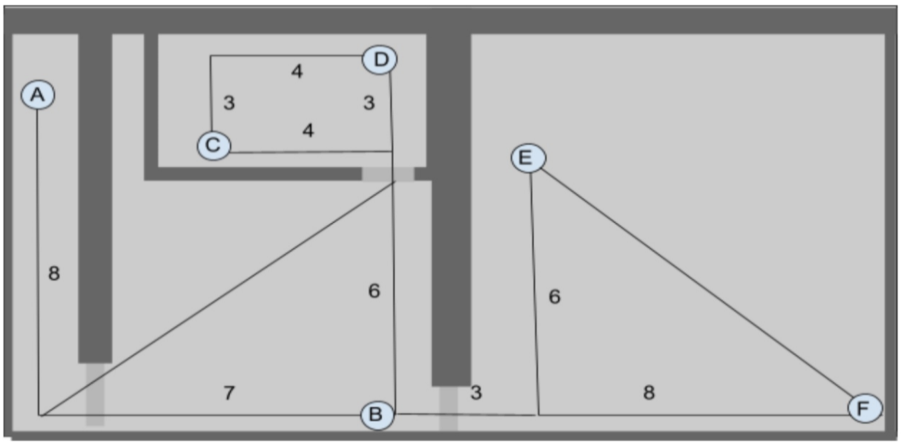
Depending on the number of trash cans or paths available, the complexity of buildings is determined. Among the three plans, the third one has the most complex structure, while the first one has the simplest structure.

With those three floor plans, three algorithms - Dijkstra's algorithm, Bellman-Ford's algorithm, and Floyd's algorithm-were set to find the shortest path. Since the distances between trash cans or the weight of edges are provided to Dijkstra's algorithm, Bellman-Ford's algorithm, and Floyd's algorithm, they will all give the same shortest path as result. Therefore, the effectiveness of those algorithms will be evaluated based on the calculation time.

[Table 1] Calculation Time (sec) of the Algorithms in Different Floor Plans

Floor Plan	Vertices	Edges	Dijkstra's	Bellman-Ford	Floyd
1	30	50	0.051s	0.053s	0.076s
2	30	100	0.071s	0.075s	0.103s
3	100	100	0.093s	0.099s	0.511s

Apart from Dijkstra's algorithm, Bellman-Ford's algorithm and Floyd's algorithm, distances between trash cans/weight of edges are not given as input to A* algorithm and RRT* algorithm. However, they work in a two-dimensional plane that has obstacles. Therefore, as A* algorithm and RRT* algorithms are different from Dijkstra's algorithm, Bellman-Ford's algorithm, and Floyd's algorithm, they were evaluated based on their ability to find the total distance shortest path.



[Figure 4] Simplification of the Building

[Table 2] Total Distance of Shortest Path and Calculation Time of A* and RRT* Algorithms

	Point	A*	Time	RRT*	Time
1	A → B	14.41	0.083s	16.13	158.28s
2	A → D	19.48	0.12s	-	> 3min
3	A → E	22.31	0.11s	-	> 3min

For Dijkstra's algorithm, Bellman-Ford's algorithm, and Floyd's algorithm, if the complexity of the structure of buildings improves by adding more vertices and edges, then deriving the conclusion from the evidence would be possible. For example, if the number of vertices and edges goes over 5000, then the gap of calculation time between algorithms will be more reliable and clear.

For A* algorithm and RRT* algorithm, if more different types of buildings are applied, then obtaining more accurate data will be possible. Also, since RRT* algorithm takes over three minutes to find the shortest path, the RRT* algorithm might be replaced with a more efficient algorithm.

6. RESULTS AND POTENTIAL CONTRIBUTION

By applying three different floor plans into Dijkstra's algorithm, Bellman-Ford's algorithm, and Floyd's algorithm, if more repetitions are done, and more data are fed, then the result will change.

For Dijkstra's algorithm and Bellman-Ford's algorithm, they both will not have dramatic changes because their calculation time will increase with the number of vertices and edges. However, for Floyd's algorithm, its calculation time will highly increase as the number of vertices increases.

For A* algorithm, more complex building structures are applied, more accurate distances are measured, and depending on the obstacles and complexity of the building structure, the result will also change.

For RRT* algorithm, if a more complex building structure is applied, then it will have a longer calculation time. So, this algorithm might not be tested later.

Since deducing the conclusion from only three sets of data is not reliable, by applying more data, determining which algorithm is applicable for the shortest path problem for trash cans in the building will be possible.

According to table 1, Dijkstra's Algorithm had slightly better calculation time results than Bellman-Ford's algorithm. Also, as the number of vertices and edges increased, calculation time increased consistently, not dramatically. For Bellman-Ford's algorithm, if more data are provided, then it might show better results than Dijkstra's. However, the results partially proved that Bellman-Ford's algorithm is almost as good as Dijkstra's and could be used in simple buildings. For Floyd's algorithm, as expected, it hugely depends on the number of vertices. So, if the building has a complex structure, Floyd's algorithm is a bad choice.

According to table 2, the A* algorithm consumes more time than algorithms using graph theory, but it found a shorter path than RRT* algorithm. For the RRT* algorithm, finding the shortest path took over three minutes. Also, it found a longer path than the path shown in the simplification of the building.

Therefore, among the algorithms using graph theory, Dijkstra's Algorithm had the shortest calculation time, consistently found the shortest path. However, Dijkstra's Algorithm can be improved by having more accurate distances between the vertices, which can be found in A* algorithm.

Although A* Algorithm consumes more time than Dijkstra's algorithm, it finds the shortest distance between trash can; hence, it is possible to provide the shortest distance.

7. REFERENCES

[1] 26258821, 3843476. "Python Implementation of RAPIDLY-EXPLORING Random Tree (RRT) Path-Planning Algorithm." Gist, 2019, gist.github.com/Fnjn 58e5eaa27a3dc004c3526ea82a92de80.

[2] [image] Bot, Jekyll. Rapidly Exploring Random Tree (RRT), 24 Apr. 2020, imgur.com/9aCP9fo.png.

[3] Han, Litao, et al. International Journal of Geo-Information, 2020, pp. 1–24, ISPRS Int. J. Geo-Inf. 2020, 9, 46; Doi:10.3390/ijgi9010046 Www.mdpi.com/Journal/Ijgi Article An Efficient Staged Evacuation Planning Algorithm Applied to Multi-Exit Buildings.

[4] Kode, Just. "Python 으로 다익스트라(Dijkstra) 알고리즘 구현하기." JustKode, 2020, justkode.kr/algorithm/python-dijkstra.

[5] Sapundzhi, F. I., and M. S. Popstoilov. Blagoevgrad, 2017, pp. 115–120, Optimization Algorithms for Finding the Shortest Paths.

[6] [image] Tyagi, Neelam “Marked Node E as Visited.” Dijkstra’s Algorithm: The Shortest Path Algorithm, 14 Dec. 2020, lh4.googleusercontent.com/78QRJLKfLw3HC1DEJYVbbzBQwvhy5htha_o0_aCfaGM0jvkep BnbCoPVRY qhMg20EtQ5v6OtV4YrtfLSh7mqiua-UWncBiyc3LqoL48GWzfsUIKbsOt5FaJbD3pz-TabQdJap870.

[7] Tyagi, Neelam. “What Is Dijkstra's Algorithm? Examples and Applications of Dijkstra's Algorithm.” What Is Dijkstra's Algorithm? Examples and Applications of Dijkstra's Algorithm, 14 Dec. 2020, www.analyticssteps.com/ blogs/dijkstras-algorithm-shortest-path-algorithm.

[8] Us, Program. “플로이드-워셜(Floyd-Warshall) 알고리즘 이론과 파이썬 구현.” IT, TISTORY, 13 June 2020, it-garden.tistory.com/247.

[9] Young, E. “[Python] 최단 경로 - 벨만 포드(BELLMAN-FORD) 알고리즘 구현하기.” Velog, Apr. 2021, velog.io/@younge/Python-%EC%B5%9C%EB%8B%A8-%EA%B2%BD%EB%A1%9C-%EB%B2%A8%EB%A7%8C-%ED%8F%AC%EB%93%9CBellman-Ford-%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98.

[10] 포함 두뇌미. “파이썬 a* (a-Star) 최단 경로 찾기 알고리즘.” PROGRAMMING PER SE, TISTORY, 11 Dec. 2020, choiseokwon.tistory.com/210.