IJRAR.ORG E-ISSN: 2348-1269, P-ISSN: 2349-5138

INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS (IJRAR) | IJRAR.ORG

An International Open Access, Peer-reviewed, Refereed Journal

Pathfinding Visualizer Using Multiple Graph Algorithms

Harsh Pandey
Delhi Technical Campus,
Greater Noida, India

Ashish Kumar
Delhi Technical Campus,
Greater Noida, India

Seema Verma
Delhi Technical Campus,
Greater Noida, India

***Abstract*— In the discipline of computer science, visualizations of algorithms aid reconstruction. Teaching and learning algorithms are time-consuming and difficult process. In any field of computer technology, visualization is a useful tool for learning. An e-learning device for shortest path algorithms visualizations is defined in this work. The powerful e-learning tools allow user to create, change, and save graph shapes, as well as see how the algorithm works. By implementing Dijkstra's, DFS, BFS and A* algorithms, the theoretical usability of the specified e-studying application is demonstrated by visual aid technology. The preliminary test results show that the e- studying tool is usable and has the potential to help college students construct useful mental models for quickest path algorithms. This e-studying application is meant to merge exceptional algorithms for determining and understanding the quickest route.**

Keywords*— Visualization, algorithms, Dijkstra's algorithm, BFS, DFS, A*, e-studying tool, quickest route.

I. Introduction

Pathfinding is the process of finding the most efficient route between two points on a map. It's a hot topic in AI research, having applications in disciplines like GPS, real- time strategy games, robotics, and logistics, and it can be applied in static, dynamic, or real-world contexts. Recent advancements in pathfinding have resulted in more enhanced, accurate, and faster solutions, and it continue to attract researchers' attention when more complicated challenges occur or new AI methods are invented. Since the publication of the Dijkstra algorithm in 1959[1], a considerable deal of research has been done in path finding to generate new algorithms that are quick and deliver the best path. Experimental data is used to validate the majority of the studies. Because experiments are highly volatile, the research must produce dependable and precise information.

The shortest path problem is strongly connected to pathfinding; consequently, pathfinding [2,5,11] is defined as determining the optimal route in a given graph (G) from a start node(s) to an end node (g), where optimal refers to the quickest route, lowest cost route, fastest path, or any other provided criteria. Pathfinding is separated into two categories: SAPF (Single Agent Pathfinding), which generates a path for a single agent, and MAPF (multi - agent Pathfinding), which generates a route for multiple agents. The single-agent pathfinding problem is exclusively considered in this work in a static environment, which implies the map does not change while the agent moves. Pathfinding has applications in a variety of sectors, and because it is difficult to explore all of them in this study, only video game applications and 2D settings are considered. For the implementation of path finding algorithm A* is proved to be very efficient algorithm, many researchers have done work for the improvement and applications of A* algorithm [3,4]

In this work various path finding algorithms are implemented to provide visual aid for the researchers, educators and students, like; Dijkstra, DFS, BFS, A*. Background the work is given in second section; Implementation is described in third section followed by result analysis.

II. Background

A literature review is necessary to evaluate issues that have not been addressed in previous studies. Many researchers attempt to understand various types of conclusions, and a literature study is required to improve earlier results. The current literature contains a variety of intriguing elements that serve as the foundation for the research and discussion.

An interesting area of mathematical concept is the quantitative study of the origin of abstract interactions among entities using graphs (networks). Although the study of these structures is completely theoretical, it can be utilized to simulate pairwise connections in a variety of real-world systems. The determination of shortest paths is one of the most widely used applications in a variety of practical uses, including graphs, industrial automation, 3D modelling, TeX formatting, urban traffic plans, best possible routing of Digital circuits, functions in complex methods, telecom controller management, broadband service navigation, estimating basic linearization, site adhoc networks.

A. *Data Structures*

Adjacency lists and adjacency matrices are the most common data structures used to describe graphs in practice. Both data structures are arrays with vertices as indexes. Here is the brief overview of the same.

The adjacency list is still the most used data structure for maintaining the graphs. A set of sets, every having one of it's vertices' peers, is termed an adjacency list. Unguided networks save each line uv two times, once within u's neighboring node and now in v's neighboring node; directed graphs get each side uv only once, inside the tip u's neighbor nodes. For both types of graphs, An entire area needed for just the adjacency list is O(V + E), where V is the vertices and E is the edges.

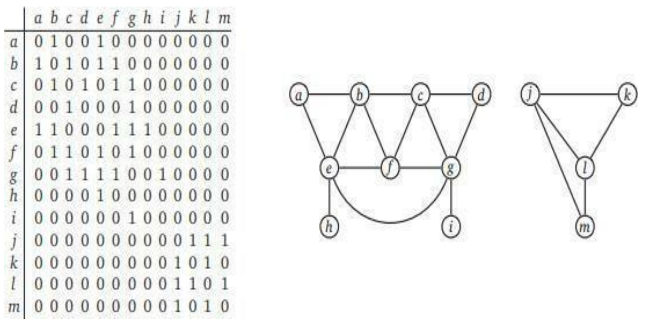
These neighbour lists can be represented in a variety of ways, but conventional adoption employs a basic one sided linked list. It can list the (out-) neighbors of a node v in O(1 + deg(v)) duration by scanning v's neighbour list. Similarly, reading the neighbour list of u can detect whether uv is a side in O(1 + deg(u)) time. It may reduce a time to O(1 + min (deg(u), deg(v))) for undirected graphs by searching the neighbour lists of u and as well as v at the same time, halting either find an edge or reach the end of the list.

The adjacency matrix was first presented by Georges Brunel in the 18th century, is the other common data structure for graphs. The adjacency matrix of a graph G is a 0s and 1s, V * V matrix, which is frequently described by a two - dimensional array A[1... V, 1... V], for each member denoting whether a specific branch is existing in G. A[u, v]:= 1 if and only if uv is E if the graph is unguided, and A[u, v]:= 1 if and only if uv is E if the graph is guided for all points u and v.

The adjacency matrix for undirected networks is always equal, meaning A[u, v] = A[v, u] for all points u and v. Directed graphs may or may not have a uniform adjacency matrix, and the diagonal elements may or not be zero.

By just searching in the right slot in an adjacency matrix, we may determine whether two points are joined by an edge in (1) time. It can scan the corresponding row to get a list of all the neighbours of a point in (V) time (or column). Even if a point has some neighbours, that still have to search the whole row to identify them all, hence this running time is ideal in the worst situation.

Adjacency matrices, on the other hand, require (V2) space independent of the number of edges in the graph, hence Only for complex graphs, they were just space-evident.



	a	b	c	d	e	f	g	h	i	j	k	l	m
a	0	1	0	0	1	0	0	0	0	0	0	0	0
b	1	0	1	0	1	1	0	0	0	0	0	0	0
c	0	1	0	1	0	1	1	0	0	0	0	0	0
d	0	0	1	0	0	0	1	0	0	0	0	0	0
e	1	1	0	0	0	1	1	1	0	0	0	0	0
f	0	1	1	0	1	0	1	0	0	0	0	0	0
g	0	0	1	1	1	1	0	0	1	0	0	0	0
h	0	0	0	0	1	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	1	0	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	0	1	1	1
k	0	0	0	0	0	0	0	0	0	1	0	1	0
l	0	0	0	0	0	0	0	0	0	1	1	0	1
m	0	0	0	0	0	0	0	0	0	1	0	1	0

Fig. 1. Adjacency Matrix

B. *Algorithms Used*

Many heuristics have been developed for AI algorithms for find paths [6,7,8,9]. There are several different ways to explore graphs. The most widely used approach is the BFS algorithm. BFS is a graph traversal approach in which start at a particular location (source or starting node) and go layer after layer through the network, studying the surroundings (nodes, which are directly connected to source node). After that, it must start moving on to the next level of neighbour nodes. It should travel the graph in a breadthwise direction, as the title suggests: Begin by travelling horizontally and examining all of the nodes in the following frame. Continue on to the next level.

The DFS algorithm, which is a programming technique, uses backtracking. It necessitates exhaustively exploring every connection, traveling further if allowed, and retracing if required. When travelling ahead and there are no other nodes along the present course, then backtrack on a same route to locate nodes to traverse. All nodes would be explored on the present course until all of the unvisited nodes have been visited, after which a new route will be chosen. The recursive behaviour of DFS can be achieved using stacks. The following is the basic concept:

Pick a starting node and stack all of its neighbours. Pop a node from the stack and put all of its neighbouring nodes into a stack to get to the next node. Continue this process until the stack is totally depleted. Ensure the nodes that travel are labeled, though. This should prevent from visiting the same node multiple time. If it doesn't label the nodes visited and back to the same node several times, that may find itself in an endless loop.

Dijksta's Algorithm[1, 6,10] begins by analyzing the graph to identify the quickest route among the root node and else such nodes in the network. This algorithm maintains a record of the quickest route among every node and the root node, and it revise these measures when a better route is found. When the quickest path among a source node and another node is found, that node is labelled as "reviewed" and joined to the route. The step has followed until route covers every other graph's node. As an outcome, it has a route that links up the root node to that of other nodes in the most efficient manner.

The A* [2,3,4] Method is a path-finding algorithm that determines the shortest route among two points.

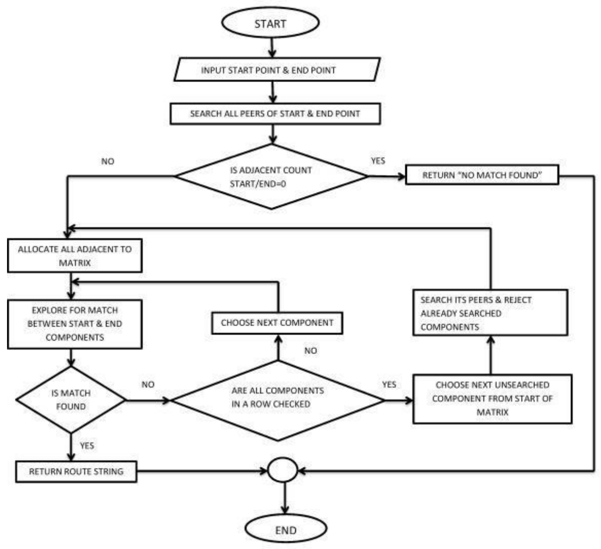

START
INPUT START POINT & END POINT
SEARCH ALL PEERS OF START & END POINT
NO
YES
IS ADJACENT COUNT START/END=0
RETURN "NO MATCH FOUND"
ALLOCATE ALL ADJACENT TO MATRIX
EXPLORE FOR MATCH BETWEEN START & END COMPONENTS
CHOOSE NEXT COMPONENT
SEARCH ITS PEERS & REJECT ALREADY SEARCHED COMPONENTS
NO
IS MATCH FOUND
NO
ARE ALL COMPONENTS IN A ROW CHECKED
YES
CHOOSE NEXT UNSEARCHED COMPONENT FROM START OF MATRIX
YES
RETURN ROUTE STRING
END

Fig. 2. Flowchart of Pathfinding(General)

When traversing a map, this is a handy algorithm for selecting the optimal route. A* was created to help with the construction of a robotic that really can discover its own Direction. It's really a useful approach for navigating graphs. It prefers shorter paths, make it a complete and efficient algorithm. An efficient algorithm will seek the most cost-effective workable solution, whereas a full algorithm will locate all possible solutions. Another feature that makes it so powerful is the usage of weighted graphs in its implementation. In a weighted graph, the cost of each set of lines is represented by numbers. This means that the algorithms can figure out the quickest and most optimal path in terms of travel and duration

III. Visualization Methodology

In this work the various algorithms are implemented by using visualization aid so as to understand it in a better way. The entire operation of software is discussed in this section. How the project began, how it functions, and how many phases of the project were completed, as well as the problems encountered at each level.

This project searches for the most direct route among start point and end point. This application will find the quickest path among start point and end point and display the whole string of each router's id among the two points. When there are many least hop count routes among the start point and end point, the first match route is chosen (Fig.2).

1. Search all the adjacent of start & end point.

2. If no adjacent for either start or end point found, go to step10.

3. Enter all adjacents row wise, separate for start & end point.

4. Search for common component among start & end point row entities.

5. If match found, go to step 9 else go to step 4 until all combinations are checked.

6. Select next unsearched component from start of matrix & discover its peers.

7. Reject already searched adjacents.

8. Go to step 3.

9. Return path string.

10. Return no match found.

11. End.

A pathfinding algorithm's primary goal is to determine the quickest route among two places. This software shows multiple pathfinding algorithms in act. All of the algorithms on this software have been converted for a two-dimensional grid, with a "cost" of 1 for 90 degree turns and 1 for moves from one node to another.

Various Stages of the work: The work has been divided into five phases for development.

All of the project's steps are covered in these phases, from data gathering to processing to user output. The five phases are as follows:

1. Graph Matrix Construction
2. Walls and event listeners have been added.
3. Integrate the Graph Algorithms.
4. Path-finding functionality has been integrated.
5. Improved the design and user interface.

Following are modules for the visualization of various algorithms on the tool:

A. Algorithm Selection

From the "Algorithms" drop-down menu, select an algorithm (Fig.3). It's worth noting that certain algorithms are unweighted while some are weighted. Weighted algorithms consider turns and weight nodes, but unweighted algorithms do not. Furthermore, not all algorithms guarantee the quickest path.

Dijkstra's Algorithm: The founder of pathfinding algorithms; ensures the quickest route.

A* Algorithm: The A* Method is a path-finding algorithm that determines the quickest route among two points.

Unweighted Breath-first Search: A fantastic approach is proposed that always takes the shortest possible route.

Unweighted Depth-first Search: Pathfinding Method has been proposed that does not always locate the quickest route.

B. Introducing up barriers

Barrier (or wall) is used to set the path for a particular area of pathfinding. To add a wall, click on the grid. A route cannot pass through walls because they are impenetrable. To visualize algorithms and perform other tasks, navbar buttons are used. From the navbar, User can erase the chosen path, barriers and obstacles, the rest of the board, and the visualization pace.

The options in the pathfinding algorithm's bar at the top are as follows (Fig. 3):

A. Algorithms mentioned

These techniques in the bar at the top were chosen based on their relevance and level of complexity. Theoretically, students have a hard time grasping these algorithms. Users will be able to comprehend these algorithms easier after they see how they are visualized. After the showcase, the user will be able to distinguish among the functionality of various algorithms based on their temporal difficulty.

Fig. 3. Example of a figure caption. (*figure caption*)

B. Mazes and Patterns

To ensure a better and clear comprehension of algorithms, a maze and patterns are added. Because there will be walls or obstructions seen between starting node and the objective node, the representation can be compared to a real scenario. Also, based on algorithm computation time, the user will be able to choose which method is better. These fun-filled options may prove to be the most appropriate way for individuals looking for a lighthearted manner to comprehend these hard topics.

C. Speed

The application has a speed bar for maintaining visualisation pace; this feature is provided so everyone understands at a varying rate, thus the user can adjust the visualisation speed to his or her preference.

D. Designing

Every node is portrayed by a matrix. Initially, a computer-generated starting and ending node will be shown. The user can move the start and end nodes around to fulfil his or her preferences.

IV. Results and Analysis

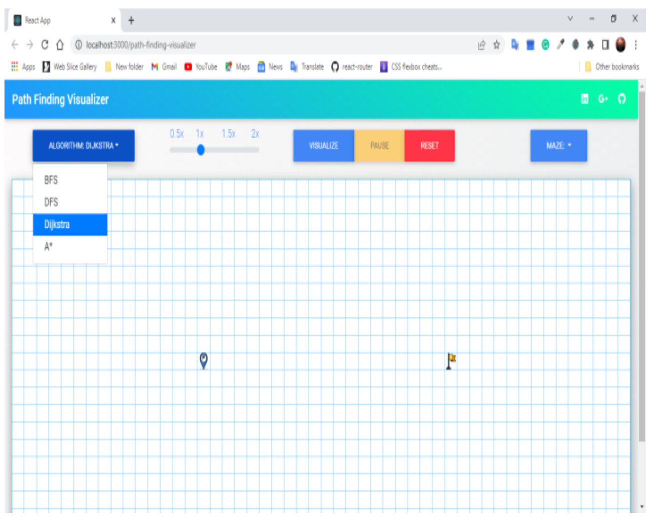
Various path finding algorithms (Dijkstra, DFS, BFS, A*) were implemented with VSCode using React JS, HTML,CSS.

Dijkstra's technique (Fig.4) is modified by the A* (A-Star) approach, which determines the quickest route faster. When it comes to determining the quickest route, the A* pathfinding technique is undoubtedly the best. Everyone uses A* as the golden key, or industry standard.

The purpose of a Breadth First Search is to search in all directions equally until the goal is accomplished. In other words, it evaluates all of the neighbours of a particular node, then all of the neighbours of the neighbours, and so on.

BFS (Fig.5) is the absolute opposite of depth-first search (DFS), which examines the node branch as deep as feasible before having to retrace.

The DFS (Fig.6) is just as effective as the BFS for generating topographical sequencing, mazes, traversing trees in a certain sequence, building tree structure, discovering a solution route with decision - making choices, detecting a loop in a network, and so on.



Depth First Search explores by travelling as far down each route as feasible prior returning. So for that reason that this technique is also referred to as Backtracking. Additionally, this trait enables the method to be constructed in both repetitive and recursively ways in a concise manner.

While Dijkstra's Algorithm is effective at determining the quickest route, it wastes time researching in directions that aren't fruitful. A* (Fig.7) enhances this by permitting additional data to be included in the heuristic function, which the algorithm can use: Dijkstra's Algorithm use the distance from the root node.The A* algorithm uses both the actual distance from the root and the estimated distance to the goal.

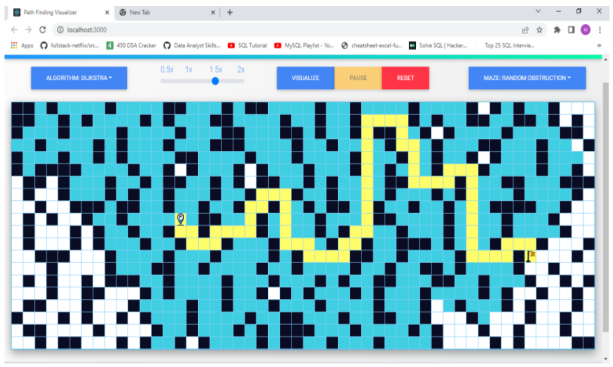


Fig. 4. Visualization of Dijkstra's Algorithm

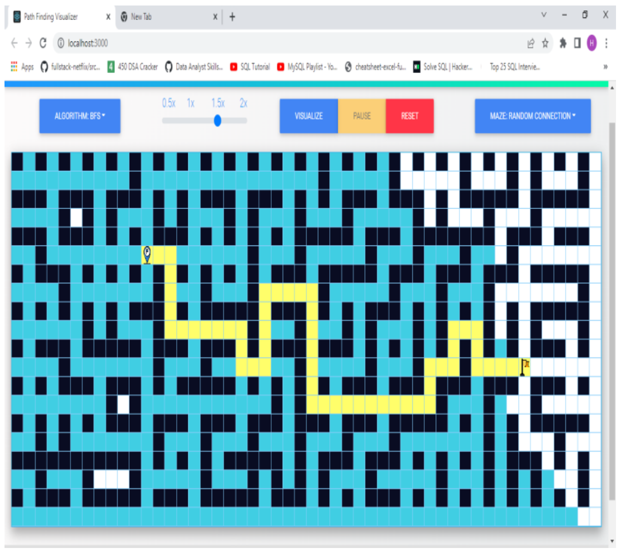


Fig. 5. Visualization of BFS Algorithm

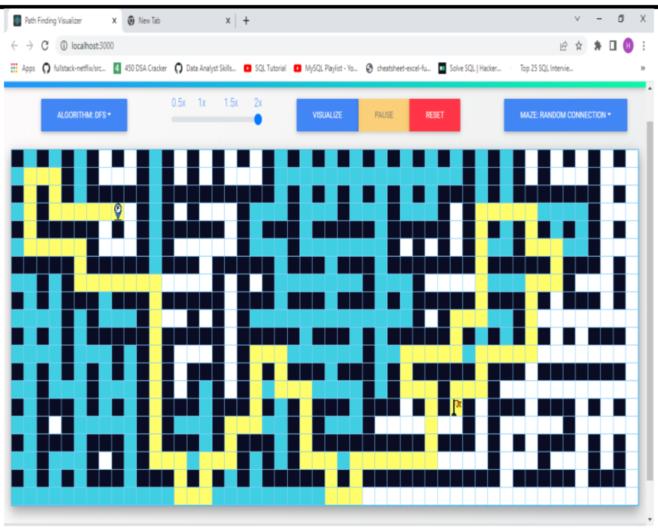


Fig. 6. Visualization of DFS Algorithm

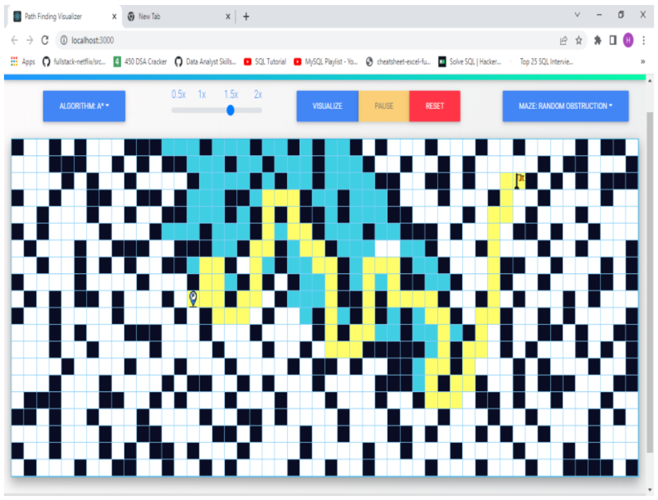


Fig. 7. Visulaization of A* Algorithm

TABLE I. COMPARATIVE ANALYSIS OF VARIOUS ALGORITHMS

Algorithm	*Time (sec:ms)*
BFS	06:46
Dijkstra	09:90
DFS	12:65
A*	04:20

Table 1 shows the time taken by various traversal algorithms on the visualization aid tool. The aim of the software is to provide solutions to various applications, mentioned below:

- It is being used to understand about algorithms using electronic mean.
- It is being used to determine the shortest route. It can be a component of the telephone network.
- In IP routing, it is used to determine the Open Shortest Possible Route First.

- It is being used to locate the vertices of a graph in rural / urban maps.
- It can create a GPS system that will direct to the appropriate sites.
- BFS indexes are built using search engine crawlers. It detects all links on the original page in order to generate new pages, starting with the source page.
- BFS is being used to find every neighbouring nodes in peer-to-peer platforms such as BitTorrent.

V. Conclusion

The work is done to develop a visualization aid for demonstrating various path finding algorithms. It achieved our goal of merging Graph Route Searching with Visual and evaluating its overall quality. There's been a major difference among conceptual and practical comprehension of algorithm implementation, as there has been in most other teaching areas. Various shortest path algorithms like; Dijkstra algorithm, DFS, BFS and A* are visually demonstrated by the designed GUI. This work aimed at providing the facility to have researchers, educators and students to demonstrate their applications with various path finding algorithms and utilize it to teach and study current combinatorial graph algorithms.

References

[1] Dijkstra, E.W. A note on two problems in connexion with graphs. Numerische Mathematik, 1959, 1:269–271

[2] Suryadibrata A, Young J, Luhulima R. Review of Various A* Pathfinding Implementations in Game Autonomous Agent. IJNMT (International Journalof New Media Technology).2019;6(1):43-49.

[3] Zheng T, Xu Y, Zheng D. AGV Path Planning basedon Improved A* Algorithm. 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC).2019, 1534-1538;

[4] Alani, Sameer, et al. "A hybrid technique for single-source shortest path-based on A* algorithm and ant colony optimization." IAES International Journal of Artificial Intelligence, 2020, 9(2): 356.

[5] Andiwijayakusuma D, Mardhi A, Savitri I, Asmoro T. A Comparative Study of the Algorithms for Path finding to Determine the Adversary Path in Physical Protection System of Nuclear Facilities. Journal of Physics: Conference Series. 2019;1198(9):092002

[6] Szczepanski, Rafal, and Tomasz Tarczewski. "Global path planning for mobile robot based on Artificial Bee Colony and Dijkstra's algorithms." 2021 IEEE 19th International Power Electronics and Motion Control Conference (PEMC).IEEE,2021:724-730

[7] Xu, Yan, et al. "Heuristic and random search algorithm in optimization of route planning for Robot's geomagnetic navigation." Computer Communications 154 (2020):12-17.

[8] Felner A, Li J, Boyarski E, Ma H, Cohen L, Kumar T et al. Adding Heuristics to ConflitBased Search for Multi-Agent Path Finding. International Conference on Automated Planning and Scheduling.2018:28

[9] Ko J, Lee D. Path Optimize Research used Ray-Tracing Algorithm in Heuristic-based Genetic Algorithm Pathfinding. Journal of Korea Game Society. 2019, 19(6): 83-90

[10] Mousaei, Ali, et al. Optimizing Heavy Lift Plans for Industrial Construction Sites Using Dijkstra's Algorithm. Journal of Construction Engineering and Management, 2021, 147(11), :04021160

[11] Algfoor Z.A., Sunar, M.S., &Kolivand,H. A comprehensive study on pathfinding techniques for robotics and video games. International journal of Computer Games and Technology (2015): 7.