FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

BIE 20303 ALGORITHM & COMPLEXITIES

PROJECT REPORT

PREPARED FOR:

PROF. DR ABD SAMAD BIN HASAN BASARI

PREPARED BY:

GAVIN CHONG JIA HAO (AI220279)

NURULAINA NISA BINTI SAHANUAN (AI220179)

SERENA NG YEN XIN (AI220061)

TUAN KHALIDAH SYAZWANA BINTI TUAN MOHD KASMAWI (AI220118)

25TH MAY 2025

Evaluating Shortest Path Solutions: A Performance Comparison of Dijkstra, BFS, and DFS Algorithms

Gavin Chong Jia Hao
Fakulti Sains Komputer dan Teknologi Maklumat
Universiti Tun Hussein Onn Malaysia
Johor, Malaysia
ai220279@student.uthm.edu.my

Nurulaina Nisa Binti Sahanuan
Fakulti Sains Komputer dan Teknologi Maklumat
Universiti Tun Hussein Onn Malaysia
Johor, Malaysia
ai220179@student.uthm.edu.my

Serena Ng Yen Xin
Fakulti Sains Komputer dan Teknologi Maklumat
Universiti Tun Hussein Onn Malaysia
Johor, Malaysia
ai220061@student.uthm.edu.my

Tuan Khalidah Syazwana Binti Tuan Mohd Kasmawi
Fakulti Sains Komputer dan Teknologi Maklumat
Universiti Tun Hussein Onn Malaysia
Johor, Malaysia
ai220118@student.uthm.edu.my

Abd Samad Bin Hasan Basari
Fakulti Sains Komputer dan Teknologi Maklumat
Universiti Tun Hussein Onn Malaysia
Johor, Malaysia
abdsamad@uthm.edu.my

Abstract—**This project investigates and compares three well-known graph traversal algorithms, Dijkstra's Algorithm, Breadth-First Search (BFS), and Depth-First Search (DFS), in the context of solving the shortest path problem. Using real-world distance data between eight key landmarks in Johor, Malaysia, each algorithm was implemented and evaluated based on performance metrics such as execution time, path accuracy, and algorithmic complexity. While all three algorithms produced the same path accuracy, the BFS algorithm delivered the fastest execution time at 28ms, followed by Dijkstra's Algorithm at 33ms, and DFS at 45ms. These findings suggest that although Dijkstra's algorithm ensures path optimality in weighted graphs, BFS may offer superior performance in small-scale or unweighted scenarios. The results offer valuable insights into algorithm selection for practical pathfinding tasks in navigation and routing systems.**

Keywords—Shortest path problem, Dijkstra's algorithm, Breadth-First Search algorithm, Depth-First Search algorithm

I. Introduction

Algorithms are essential tools that enable efficient problem-solving. Particularly in the field of graph theory, different algorithms are designed to traverse and analyse networks of nodes and edges. One common problem in graph theory is the shortest path problem, which seeks to find the most efficient route between two points in a network. This problem has wide applications, from GPS navigation systems to network routing and logistics optimization. There are several algorithms available to solve this problem, each with its strengths and trade-offs.

This project focuses on three classic algorithms—Dijkstra's Algorithm, Breadth-First Search (BFS), and Depth-First Search (DFS)—each representing different approaches to graph traversal. Dijkstra's Algorithm is a greedy algorithm that computes the shortest path tree from a single source node in a graph with non-negative edge weights to solve the shortest path problem [1]. Breadth-First Search is a graph traversal technique that explores all nodes at one depth level before progressing to nodes at the next level [2]. Depth-First Search, in contrast, uses a deep traversal approach, by exhaustively traversing a single branch before backtracking to previously unexplored nodes [3].

The objective of this study is to implement these three algorithms, apply them to real-world location data, and compare their outputs and performance. By analysing the algorithms in terms of their time complexity, traversal logic, and resulting paths, this project aims to determine the most efficient solution for different use-case scenarios of the shortest path problem.

II. Background

A. Algorithm and Complexity

The effectiveness of an algorithm is measured not only by its ability to solve the problem, but also by its computational complexity. The amount of resources such as time and memory each algorithm consumes is taken into consideration. The upper bound of an algorithm's growth rate is expressed using Big-O notation. Additionally, best case, average case, and worst case analysis is done on each algorithm to compare resource usage and evaluate the algorithm's efficiency. Best case represents the scenario in which the algorithm performs the minimum number of steps; average case represents the expected performance over all possible inputs; worst case represents the scenario in which the algorithm requires the maximum number of steps or memory.

These analyses provide a comprehensive understanding of an algorithm's behaviour and are essential for comparing alternatives. In this project, Dijkstra's Algorithm, Breadth-First Search (BFS), and Depth-First Search (DFS) were analysed and implemented to determine their computational complexities. The time taken by each algorithm to compute the shortest path was recorded and compared, alongside an analysis of how the algorithms scale and perform under the specific constraints of the dataset.

B. Shortest Path Problem

In the context of this project, the shortest path problem was modeled using a graph of eight landmarks across Johor, Malaysia. Each node in the graph represents a location (e.g., LEGOLAND Malaysia, Forest City, Puteri Harbour), and each edge represents a direct travel route between two locations, with the distance used as the edge weight. By applying different algorithms to a geographical dataset of Johor, the influence of different traversal methods on route planning can be simulated for practical navigation systems.

III. Literature Review

In order to solve real-world problems like transportation planning, the shortest path algorithms are essential. Dijiktra's algorithm, Breadth-First Search (BFS), and Depth-First Search (DFS) are some of the most extensively researched algorithms as each has unique benefits based on the needs and structure of the application. To ascertain the accuracy, scalability, and efficiency of these algorithms, numerous researchers have investigated and assessed their performance in various scenarios. Table 1 summarizes a few research articles that used Dijkstra, BFS, and DFS, emphasizing their application areas and significant conclusions to help readers better grasp how well these algorithms work across different domains.

Table 1: Comparison of Research Findings

Authors and Years	**Algorithms Used**	**Application Domain**	**Main Findings**
Elkari et. al (2024) [4]	DFS, BFS, A*	Pathfinding in maze environments for learning, analysis, and performance comparison.	A* consistently outperformed DFS and BFS in terms of both path cost and execution time. BFS ensures completeness but slower, while DFS is memory-efficient but less accurate.
Aliyan et al. (2024) [5]	DFS, BFS, A*, Dijikstra, Bellman-Ford	Broad application in network routing, AI, and computational models.	A* was the most efficient when a well-designed heuristic was applied. Dijikstra' algorithm remained optimal for non-negative weighted graphs, and Bellman-Ford was particularly suitable for handling negative edge weights.
Weibel et al. (2020) [6]	DFS, BFS, Dijikstra, A*	Demonstrating algorithm behavior in 2D grid games and pathfinding visual tools.	A* yielded the most favorable overall performance in terms of speed and path quality. Dijikstra was dependable and accurate, while BFS was optimal in unweighted scenarios. DFS proved to be inefficient and unsuitable for shortest path applications.
Susanto et al. (2021) [7]	BFS, Dijikstra, A*	Simulates pathfinding with traffic-related variables (weather, obstructions) on maps.	Dijikstra's algorithm achieved the fastest computation time, outperforming A* by 13.78% and BFS by 32.3%. However, A* produced more optimal paths due to heuristic integration, while BFS lagged in both speed and path accuracy.
James & Chandran (2020) [8]	DFS, BFS, Dijikstra, A*	Evaluates resource-efficient pathfinding for real-time navigation (CPU, memory, etc.).	BFS is the fastest algorithm in time. A* and Dijkstra is the most accurate algorithm. As for DFS, it was found to be the least optimal in both performance and accuracy.
Elsahhed et al. (2025) [9]	A *, BFS, DFS, Dijikstra, Best-First + ILS	Focuses on high-efficiency grid navigation in resource-constrained or real-time environments.	ILS significantly enhanced performance, reducing execution time by 87% and node expansions by over 70% while maintaining near-optimal path quality across various grid environments.
Permana et al. (2018) [10]	A*, Dijkstra, BFS	Optimizing NPC navigation in 2D grid games with player interaction and obstacle placement.	All algorithms produced equally short paths, but A* required fewer computations and less memory. Dijikstra had the shortest execution time in two out of the three tests, while BFS consistently processed more nodes, making it less efficient in a constrained environment.

IV. Methodology

This section outlines the algorithms selected to solve the shortest path problem, justifies their selection, describes the dataset used, and presents the assumptions and constraints considered during the implementation. The methodology plays a key role in ensuring that the project is built on logical reasoning and empirical comparison.

A. Description of Algorithms Used

In a weighted graph with non-negative edge weights, Dijkstra's Algorithm is a classic greedy algorithm that effectively determines the shortest path from a single source node to all other nodes, maintaining a priority queue to always explore the node with the shortest tentative distance [11]. This algorithm is perfect for our dataset because all distances between locations are positive, making it appropriate for dependable and optimal shortest path computations.

Breadth-First Search (BFS) is an algorithm used for traversing graphs, which examines all adjacent nodes at the current level before proceeding to the next one [12]. Although BFS is generally applied to unweighted graphs, it is included in this analysis to evaluate its effectiveness and path-finding capabilities on our dataset, where all edge weights are considered equal. This approach illustrates the variation in results when distance metrics are disregarded.

In contrast, Depth-First Search (DFS) investigates a branch as far as it can go before turning around [13]. Similar to BFS, DFS is helpful for examining graph structure but does not take edge weights into consideration. The pathfinding behavior of DFS is examined in this study, along with how it varies from BFS and Dijkstra's in terms of performance and applicability for shortest path finding.

B. Justification of Chosen Algorithms

The selection of Dijkstra's Algorithm, BFS, and DFS was based on the desire to evaluate algorithms from different categories—greedy search, uninformed traversal, and depth-based exploration—and how they approach the shortest path problem.

Because of its accuracy and effectiveness on graphs with non-negative weights, Dijkstra's technique was selected as the main technique for calculating the shortest path. It is extensively utilized in real-world navigation systems, including transportation networks and route planning, and it ensures the best route [14].

BFS was used as an example of an unweighted method. It is efficient in terms of time complexity and ensures the shortest path in terms of the number of edges, even if it ignores distance values [15]. This enables us to compare the effects of treating all pathways equally (ignoring weights) and observe its behavior on the same dataset.

In order to assess how well it performs in path finding and graph navigation, DFS was included. Its presence provides a useful difference in algorithm design and behavior, even though it is not appropriate for shortest path computation in weighted or even unweighted networks [13]. We can demonstrate why certain algorithms are more suited for shortest route issues than others by using DFS.

C. Dataset Description

The dataset used in this project represents a weighted undirected graph, where each node corresponds to a specific location in Johor, Malaysia, and each edge represents the road distance (in kilometers) between two locations. A total of eight locations are represented as nodes labeled from A to H, and 28 weighted connections represent the distances between them.

The locations included are as follows:

A – LEGOLAND Malaysia

B – Forest City

C – XPark Sunway Iskandar

D – Puteri Harbour

E – Taman Botani Iskandar

F – Johor Bahru Chinese Heritage Museum

G – Galeri Seni Johor

H – The Malay Cultural Village

Some examples of the connections include A–E (3.8 km), C–D (7.3 km), and G–H (7.2 km). All roads are assumed to be bidirectional with symmetric weights, meaning the distance from A to B is the same as from B to A. The dataset provides a real-world context for evaluating the performance and results of the algorithms.

D. Assumptions and Constraints

Several assumptions and constraints were made to guide the design and evaluation of the algorithms. Firstly, the graph is considered undirected and connected, ensuring that there is at least one path between any pair of locations. All edge weights are positive and static, meaning they do not change during the algorithm's execution.

The algorithms assume that the dataset accurately reflects the actual distances between locations and that no external factors, such as traffic conditions or road closures, affect these values. For BFS and DFS, the edge weights are ignored, treating all connections as having equal cost. This assumption allows for the inclusion of these algorithms for comparison, despite their limitations in handling weighted graphs.

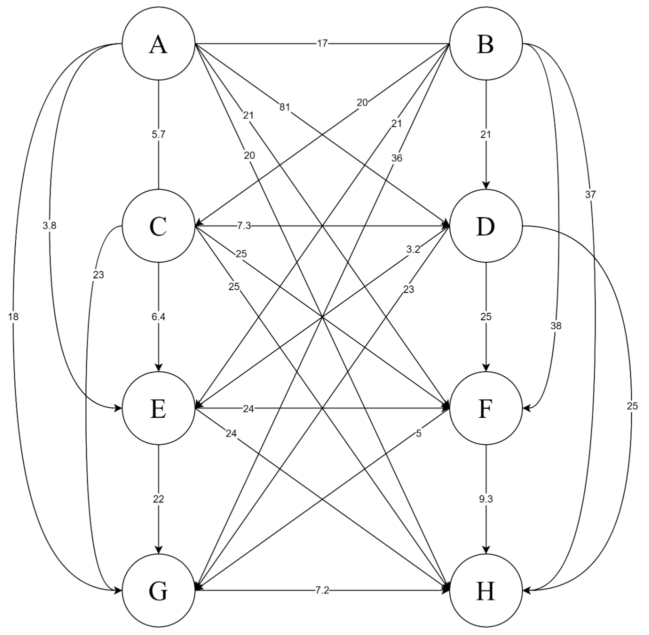
Finally, the system does not include dynamic elements or real-time updates. All computations are performed on a fixed dataset, and results are evaluated based on algorithm performance under these static conditions.

V. Implementation and Results

This section outlines the implementation of the 3 selected algorithms to solve the shortest path problem based on the chosen data set.

A. Implementation Details

The selected programming language is C++ due to its wide range of pre-written code in the Standard Template Library such as vector and priority_queue. The tool used is Visual Studio Code.



Keyword	
A	LEGOLAND Malaysia
B	Forest City
C	XPark Sunway Iskandar
D	Puteri Harbour
E	Taman Botani Iskandar
F	Johor Bahru Chinese Heritage Museum
G	Galeri Seni Johor
H	The Malay Cultural Village

The graph above shows the distance between 8 chosen locations, each node represents one location, and the graph only contains a directed path. The chosen algorithm will be applied on the dataset above to solve the shortest path problem.

B. Algorithm Descriptions and Pseudocode

Below is the pseudocode of the 3 selected algorithms based on the dataset above. Each algorithm is generated to ensure that it could solve the shortest path problem.

```
1) Dijkstra's Algorithm
   Initialize dist[] with distance between nodes
   Initialize adj[] with the adjacent nodes
   Initialize shortD[] with -1
   Initialize vis[] with false
   Create min-priority_queue pq storing pairs of
   double and int and enqueue with (0, 0)
   While pq is not empty:
      Dequeue (curr_dist, node)
      If vis[node] is true:
         Continue
      Mark vis[node] as true
      Set shortD[node] = curr_dist
      For each next in adj[node]:
         Let next_dist = curr_dist + dist[node][next]
         Enqueue(next_dist, next)
2) Breadth First Search
   Initialize dist[] with distance between nodes
   Initialize adj[] with the adjacent nodes
   Initialize shortD[] with infinity
   Initialize vis[] with false
   Create queue q and enqueue with (0, 0)
   While q is not empty:
      Dequeue (node, curr_dist)
      Set shortD[node] = min(shortD[node], curr_dist)
      If vis[node] is true:
         Continue
      Mark vis[node] as true
      For each next in adj[node]:
         Let next_dist = curr_dist + dist[node][next]
         Enqueue(next, next_dist)
3) Depth First Search
   Initialize dist[] with distance between nodes
   Initialize adj[] with the adjacent nodes
   Initialize shortD[] with infinity
   Initialize vis[] with false
   Define Function DFS(node, curr_dist):
      Set shortD[node] = min(shortD[node, curr_dist)
      For each next in adj[node]:
         Let next_dist = curr_dist + dist[node][next]
         Call DFS(next, next_dist)
   Call DFS(0, 0)
```

C. Performance Metrics and Complexity

1) Time Complexity (Step Count & Big-O)
   Let m = number of edges in the graph above
   and n = number of nodes in the graph

i) Dijkstra's Algorithm

Code	*Count*
Initialize dist[]	m
Initialize adj[]	m
Initialize shortD[]	n
Initialize vis[]	n
Enqueue priority queue	1
Outer Loop: extract-min from pq	m
Outer Loop: determine vis[node] visited or not	m
Outer Loop: mark vis[node]	n
Outer Loop: update shortD	n
Inner Loop: Enqueue adj[node]	m

Total Steps = m + m + n + n + 1 + m + m + n + n + m = 5m + 4n + 1

Time Complexity = O((m+n))

ii) Breadth First Search

Code	*Count*
Initialize dist[]	m
Initialize adj[]	m
Initialize shortD[]	n
Initialize vis[]	n
Enqueue queue	1
Outer Loop: dequeue from queue	m
Outer Loop: determine vis[node] visited or not	m

Code	*Count*
Outer Loop: mark vis[node]	n
Outer Loop: update shortD	n
Inner Loop: Enqueue adj[node]	m

Total Steps = m + m + n + n + 1 + m + m + n + n + m = 5m + 4n + 1

Time Complexity = O((m+n))

iii) Depth First Search

Code	*Count*
Initialize dist[]	m
Initialize adj[]	m
Initialize shortD[]	n
Initialize vis[]	n
Recursive Function	m
Outer Loop: update shortD	m
Inner Loop: Enqueue adj[node]	m * n

Total Steps = m + m + n + n + m + m + (m * n) = mn + 4m + 2n

Time Complexity = O((m*n))

2) Best, Average, and Worst Case Analysis
   i) Dijkstra's Algorithm
      a) Best Case
         When the first path from the starting node to each of the other nodes is the shortest path, no reassignment is needed. The Time Complexity will be O(m+n).
      b) Average Case
         When some of the new path is shorter than the previous one, the priority_queue will have to spend time to rearrange the order of the path when there is a shorter path inserted. The Time Complexity will be O(m+n).
      c) Worst Case
         When all the new paths are shorter than the previous one, the priority_queue will have to spend maximum time to rearrange the order of the path as each time a new path is inserted, the rearrangement will take place. The Time Complexity will be O(m+n).
   ii) Breadth First Search
      a) Best Case
         When the first path from the starting node to each of the other nodes is the shortest path, no reassignment is needed. The Time Complexity will be O(m+n).
      b) Average Case
         When the shortest path is in the middle of the dataset, reassignment is needed when a shorter path is found. The Time Complexity will be O(m+n).
      c) Worst Case
         When all the new paths are shorter than the previous one. The reassignment will take place each time a new path is viewed, leading to a time wastage. The Time Complexity will be O(m+n).
   iii) Depth First Search
      a) Best Case
         When the first traversal path is the shortest path, no reassignment is needed. The Time Complexity will be O(m*n).
      d) Average Case
         When the shortest path is in the middle of the dataset, reassignment is needed when a shorter path is found. The Time Complexity will be O(m*n).
      e) Worst Case
         When all the new paths are shorter than the previous one. The reassignment will take place each time a new path is viewed, leading to a time wastage. The Time Complexity will be O(m*n).

D. Results & Comparison

1) Execution Time Comparison
   i) Dijkstra's Algorithm - 33ms
   ii) Breadth First Search - 28ms
   iii) Depth First Search - 45ms

In evaluating the runtime performance of the three shortest-path algorithms, a clear variation in execution efficiency is observed. The Breadth First Search (BFS) approach recorded the fastest execution time at 28 milliseconds, followed by Dijkstra's algorithm at 33 milliseconds, while the Depth First Search (DFS) approach was the slowest at 45 milliseconds.

The superior performance of the BFS method can be attributed to its lightweight operations and simpler queue structure. Although it respects edge weights, it does not employ a priority queue, making it computationally less intensive per iteration compared to Dijkstra. Dijkstra's algorithm, while more accurate in general graphs with weighted edges, incurs additional overhead from managing a min-priority queue, slightly increasing its execution time.

The DFS approach, despite correctly computing the shortest paths, demonstrates the least efficiency due to its exhaustive traversal nature. DFS explores many redundant or suboptimal paths recursively without early pruning, resulting in higher recursive call overhead and increased processing time. This reflects the trade-off between simplicity and performance in pathfinding algorithms, especially when weights are involved.

In summary, BFS offers the best execution speed in this case, while Dijkstra provides a balanced trade-off, and DFS, although accurate, lags in speed due to its brute-force traversal style.

2) Path Accuracy / Validity
   i) Dijkstra's Algorithm - Correct Path
   ii) Breadth First Search - Correct Path
   iii) Depth First Search - Correct Path

Despite differences in algorithmic design and execution time, all three algorithms successfully computed the correct shortest paths and distances from the source node A to all other nodes in the graph. The paths and distances they generated perfectly match the reference output, confirming the correctness of their implementations.

Dijkstra's algorithm guarantees the shortest path in weighted graphs with non-negative weights using a greedy strategy, which explains its accurate results. Breadth First Search algorithm was also adapted correctly to consider the edge weights while exploring neighbors. As a result, it identified the optimal paths despite traditionally being used for unweighted graphs. While the Depth-First Search algorithm, although not typically used for shortest path in weighted graphs due to its depth-oriented approach, was enhanced by tracking and updating the minimum distances recursively. This allowed it to eventually yield correct shortest paths, albeit less efficiently.

This consistency across all algorithms shows that, from a functional correctness standpoint, each algorithm reliably computes the expected shortest paths in this graph. The differences among them therefore lie not in path validity, but in performance efficiency, recursion overhead, and queue/prioritization strategies.

3) Visual Graph Outputs (if any)

i) Dijkstra's Algorithm

```
A: distance = 0     path = A
B: distance = 17    path = A->B
C: distance = 5.7   path = A->C
D: distance = 13    path = A->C->D
E: distance = 3.8   path = A->E
F: distance = 21    path = A->F
G: distance = 18    path = A->G
H: distance = 20    path = A->H

Execution Time: 33 microseconds
```

ii) Breadth First Search

```
A: distance = 0     path = A
B: distance = 17    path = A->B
C: distance = 5.7   path = A->C
D: distance = 13    path = A->C->D
E: distance = 3.8   path = A->E
F: distance = 21    path = A->F
G: distance = 18    path = A->G
H: distance = 20    path = A->H

Execution Time: 28 microseconds
```

iii) Depth First Search

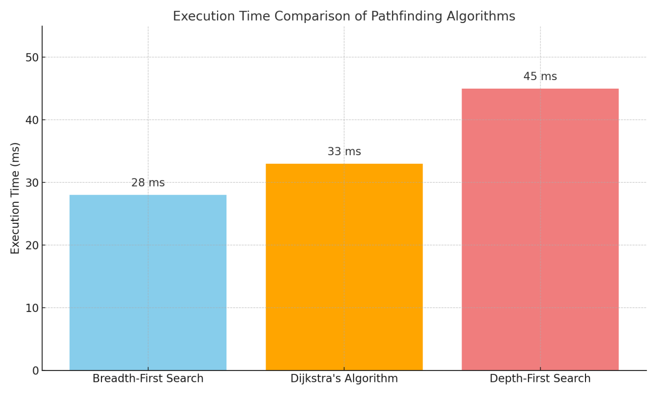
```
A: distance = 0     path = A
B: distance = 17    path = A->B
C: distance = 5.7   path = A->C
D: distance = 13    path = A->C->D
E: distance = 3.8   path = A->E
F: distance = 21    path = A->F
G: distance = 18    path = A->G
H: distance = 20    path = A->H

Execution Time: 45 microseconds
```

4) Comparative Tables and Figures

Algorithm	Comparison	
	Execution Time	***Path Accuracy***
Dijkstra's Algorithm	33ms	Correct
Breadth First Search	28ms	Correct
Depth First Search	45ms	Correct

Comparative Tables for 3 algorithms



Execution Time Comparison Figure

VI. Conclusion

This project has successfully demonstrated the practical application and comparative evaluation of Dijkstra's Algorithm, Breadth-First Search, and Depth-First Search in solving the shortest path problem using real geographical data. The implementation results show that while all three algorithms achieved the same level of accuracy in computing paths, Breadth-First Search emerged as the most efficient in terms of execution speed, completing in just 28ms, compared to Dijkstra's 33ms and DFS's 45ms.

The comparative results offer practical insights into algorithm selection for navigation, routing, and other pathfinding applications. These results indicate that BFS is highly suitable for smaller or unweighted graphs where minimising the number of steps is sufficient. Although Dijkstra's algorithm remains the most mathematically optimal for weighted graphs due to its consideration of edge costs, it incurs a slightly higher computational cost. DFS, while useful for thorough path exploration, is less efficient and not reliable for shortest path computation in practical scenarios.

The findings reinforce the importance of choosing the appropriate algorithm based on the nature of the data and the specific requirements of the problem. BFS is an excellent choice for speed in simple environments, Dijkstra's for optimality in weighted networks, and DFS for applications requiring full path exploration. This comparative study contributes valuable insight into designing efficient pathfinding solutions for navigation and routing applications.

References

[1] D. Rachmawati and L. Gustin, "Analysis of Dijkstra's algorithm and A* algorithm in shortest path problem," J. Phys.: Conf. Ser., vol. 1566, no. 1, p. 012061, Jun. 2020, doi: 10.1088/1742-6596/1566/1/012061.

[2] M. Aliyan et al., "Analysis and Performance Evaluation of Various Shortest Path Algorithms," 2024 3rd International Conference for Innovation in Technology (INOCON), Bangalore, India, 2024, pp. 1-16, doi: 10.1109/INOCON60754.2024.10512150.

[3] R. AlKahtani, A. Alhabdan, M. Alosami, W. Alshammari, A. A. Abdo and L. Hamdi, "Comparative Analysis between BFS and DFS-Shortest Path Algorithms," 2025 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 2025, pp. 1-5, doi: 10.1109/eStream66938.2025.11016860.

[4] ELKARI, B., OURABAH, L., SEKKAT, H., HSAINE, A., ESSAIOUAD, C., BOUARGANE, Y., & El Moutaouakil, K. (2024). Exploring Maze Navigation: A Comparative Study of DFS, BFS, and A* Search Algorithms. *Statistics, Optimization & Information Computing*, *12*(3), 761-781.

[5] Aliyan, M., Hasan, M. Z., Qayoom, H., Hussain, M. Z., Nosheen, S., Mustafa, M., ... & Bilal, A. (2024, March). Analysis and performance evaluation of various shortest path algorithms. In *2024 3rd*

[6] Weibel, C., Chirila, D. V., Johnson, K., Rasocha, L., Kaas-Mason, M., Špralja, R., & Reinhardt, L. (2020). Comparing Path-Finding Algorithms

[7] Susanto, W., Dennis, S., Handoko, M. B. A., & Suryaningrum, K. M. (2021, October). Compare the path finding algorithms that are applied for route searching in maps. In *2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)* (Vol. 1, pp. 178-183). IEEE.

[8] James, S. O., & Chandran, A. S. COMPARITIVE ANALYSIS ON SHORTEST PATH COMPUTATION IN GPS SYSTEMS.

[9] Elshahed, A., Ali, M. K. B. M., Mohamed, A. S. A., Abdullah, F. A. B., & Aun, T. L. J. (2025). Efficient Pathfinding on Grid Maps: Comparative Analysis of Classical Algorithms and Incremental Line Search. *IEEE Access*.

[10] Permana, S. H., Bintoro, K. Y., Arifitama, B., & Syahputra, A. (2018). Comparative analysis of pathfinding algorithms a*, dijkstra, and bfs on maze runner game. *IJISTECH (International J. Inf. Syst. Technol*, *1*(2), 1.

[11] T. A. AI, "Breadth First Search (BFS) in AI," Applied AI Blog, Jan. 28, 2025. https://www.appliedaicourse.com/blog/bfs-in-ai/

[12] geeksforgeeks, "What is Dijkstra's Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm," GeeksforGeeks, Mar. 10, 2023. https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/

[13] "Depth First Search (DFS) for Artificial Intelligence," GeeksforGeeks, May 16, 2024. https://www.geeksforgeeks.org/depth-first-search-dfs-for-artificial-intelligence/

[14] A. Singh, "Dijkstra's Shortest path algorithm," Medium, May 01, 2023. https://medium.com/@adsingh03011/dijkstras-shortest-path-algorithm-e9f06b31810

[15] "Shortest Path in Unweighted Undirected Graph using BFS," Pencil Programmer, 2023. https://pencilprogrammer.com/algorithms/shortest-path-in-unweighted-graph-using-bfs/#google_vignette (accessed Jun. 08, 2025).