Optimization of Fastest Public Transportation Route Selection in Jakarta Metropolitan Area Using Uniform Cost Search and A* Algorithm

Hanif Kalyana Aditya - 13523041
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: hanifaditya2304@gmail.com , 13523041@std.stei.itb.ac.id

***Abstract*— This paper addresses the problem of finding the fastest route through public transportation networks in the Jakarta Metropolitan Area (Jabodetabek), which includes various modes such as KRL, MRT, Trans Jakarta, and LRT. The complexity of the transportation system and the absence of an integrated route planning tool often make it difficult for passengers to identify the most efficient path. To solve this, we model the transportation system as a weighted directed graph, where nodes represent stations and edges represent travel segments with associated time costs.**

We implement two well-known graph search algorithms—Uniform Cost Search (UCS) and A* Search—to find the shortest travel time between two stations. The A* algorithm utilizes a heuristic function based on mode-switch penalties to improve search efficiency. The system allows users to choose both the algorithm and the start/goal nodes and provides a visualization of the state space tree formed during the search.

Testing shows that A* consistently reduces the number of nodes visited compared to UCS, while still producing optimal results. This demonstrates the potential of informed search in solving real-world transportation optimization problems. (*Abstract*)

Keywords—Public Transportation; Jakarta; Graph; Uniform Cost Search Algorithm; A* Algorithm

I. Introduction

Transportation plays a crucial role in the economic and social activities of urban populations. In the Jakarta Metropolitan Area (also known as Jabodetabek), the complexity and density of public transportation networks have increased significantly with the development of various modes such as KRL (Commuter Line), MRT, Trans Jakarta, and LRT. While these systems aim to improve connectivity and efficiency, users often face difficulties in identifying the fastest route across different transport modes and operators.

In everyday scenarios, determining the most time-efficient route is not a trivial task. A traveler may need to transfer between multiple stations, change modes of transport, or traverse a non-optimal path due to lack of centralized route planning. These challenges are essentially computational problems that can be modeled as a graph traversal problem, where each station is represented as a node and each route segment as a weighted edge (with travel time as weight).

This research explores how Uniform Cost Search (UCS) and A* algorithms can be utilized to solve the route optimization problem in Jabodetabek's public transportation network and focused. Both algorithms are widely known in the field of Graph Theory for solving shortest path problems and in this context will be used to find the shortest travel-time. UCS algorithm itself guarantees the optimal solution in terms of total cost, while A* enhances search efficiency through heuristics. To make the heuristic admissible yet simple, a condition where a traveler changes type of transportation to commute is used as heuristic. The reason for the usage of this heuristic is because in real-life, travelers need several minutes to switch into another type transportation.

The scope of this study includes the modeling of public transportation networks in Jabodetabek as a directed weighted graph, with sample data representing selected stations or bus stops and travel times. Besides analyzing the difference between two algorithms, the objective is to develop a functional Python program that be able to determine the fastest route between two stations with visual feedback in the form of a state space tree.

Through this paper, the demonstration of practical application of search algorithms is focused on so that it could be used to solve real-world problems in future alongside highlighting the difference between uninformed and informed search strategies in terms of performance and complexity.

II. Theoretical Foundation

A. Graph Traversal

Graph traversal refers to the process of visiting nodes in a graph structure, where the graph is defined as a collection of vertices (nodes) connected by edges. In the context of route planning, each node can represent a location (e.g., a station or stop), and each edge represents a connection or segment of the route, often associated with a weight such as distance, time, or

cost. The goal of traversal is typically to find an optimal path from a starting node to a goal node based on some criteria, such as minimum travel time.

Traversal algorithms can vary depending on whether the goal is to explore all nodes (e.g., Breadth-First Search) or to find the shortest path (e.g., Dijkstra's algorithm, UCS, or A*). In this study, traversal is used to determine the fastest route in a directed weighted graph where edges represent public transportation links with travel time as their weight.

B. *Informed and Uninformed Search*

Search algorithms are generally classified into two categories: uninformed (blind) and informed (heuristic-based).

- Uninformed search does not have any domain-specific knowledge or heuristic guidance. It explores the graph based solely on the information available in the graph structure. An example is Uniform Cost Search (UCS), which always expands the node with the lowest cumulative cost from the start.
- Informed search, on the other hand, leverages heuristic functions to estimate the cost from a node to the goal, allowing the algorithm to prioritize more promising paths. A* is a well-known informed search algorithm that uses both the actual cost from the start node and the estimated cost to the goal node to guide the search more efficiently.

Understanding the distinction between these two types of search is important when evaluating the trade-offs between completeness, optimality, and performance.

C. *Uniform Cost Search (UCS) Algorithm*

Uniform Cost Search or UCS algorithm is one of the route planning algorithms or route determination without any additional information about the search destination (Uninformed Search/Blind Search). The search is done based on the cost or cost to reach a node in the graph. Cost or cost in UCS is usually denoted as g(n), where n is a node in the UCS search graph. In the minimum cost search, the node that has the lowest cost to reach a point will be generated first.

In the cost determination process, no additional information is known about the search destination, so this algorithm is included in Uninformed Search. In this algorithm, the node that has the lowest cost will be generated first, so the implementation of this algorithm usually uses Priority Queue, where the node that has the highest priority will be at the front. The priority value of this algorithm is the same as the cost, so the following equation is used.

$$f(n) = g(n)$$

where $f(n)$ is the evaluation function that becomes the priority value of a node and g(n) is the cost value of a node.

D. *A* Algorithm*

The A* (A-star) algorithm is one of the route planning algorithms or route determination with additional information about the search destination (Informed Search). The A* algorithm is a development of the Uniform Cost Search and Greedy Best First Search algorithms, with the idea of avoiding expensive paths or nodes in addition to using heuristic values. The search is carried out based on the evaluation function (f (n)) of a node. This evaluation function is the sum of the costs to reach a node added to the heuristic value of the node to reach the destination node.

In the A* algorithm, the node with the best evaluation function value (the smallest in the case of a minimum cost search) will be generated first. In this algorithm, the node with the lowest evaluation function value will be generated first, so the implementation of this algorithm usually uses a Priority Queue, where the node with the highest priority will be at the front. The priority value of this algorithm is the same as the heuristic value of a node, so the following equation is used.

$$f(n) = g(n) + h(n)$$

where f(n) is an evaluation function that becomes the priority value of a node, g(n) is the cost value of a node, and h(n) is a heuristic value that is an estimate of the distance cost from node n to the destination node. In the A* algorithm, there is a concept of admissible heuristics, namely a heuristic function (h(n)) that is always smaller than the actual cost from node n to the destination node. If a heuristic function is used in the search, then A* is guaranteed to produce an optimal solution.

III. Methodology

A. *Public Transportation Networks*

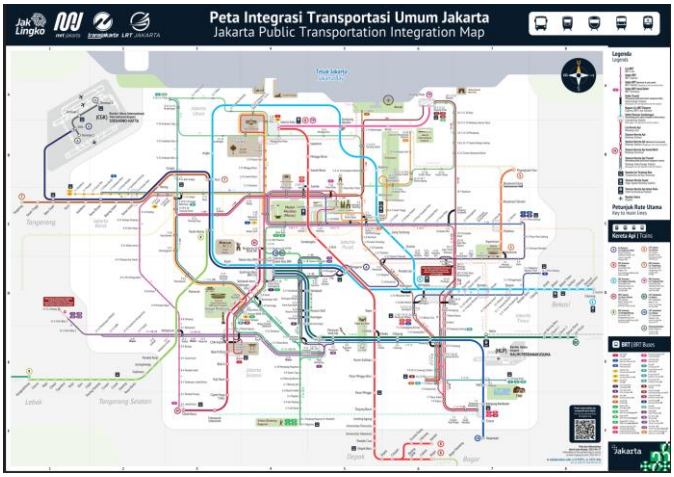

Fig. 3.1 Jakarta's Integrated Public Tranportation Map

(Source: https://transjakarta.co.id/rute)

To model transportation that system into a computational problem, we first manually collected actual route and station/bus stop data from the official websites and publicly available sources of four major transportation type—Mass

Rapid Transit (MRT), Light Rapid Transit (LRT), Trans Jakarta (busway) and subway (KRL). These sources include route maps, line information, travel time estimates, and station names.

The challenge of this problem relies on the complexity and density of the real-world Jabodetabek's public transportation full network. Because of that, only selected routes and stations or bus stops were used for this implementation. The selected stations or bus stops are generally major transit hub/point and representative routes from each transportation mode is included.

To simplify the system for processing, stations or bus stops that overlap across different modes (e.g., "Sudirman / Dukuh Atas") were grouped into a single node. Similarly, similar stations like "Blok M" and "ASEAN" were treated as one. This is because, those two places are actually transit hub which are close to each other in reality. The network was then modeled as a directed weighted graph, where each node represents a station or bus stop, and each edge represents a connection between two stations or bus stops with an associated travel time and mode of transportation. This simplification allows us to focus on computational problems without losing the essence of real-world complexity. (Table attached)

IV. IMPLEMENTATION

A. *Uniform Cost Search Implementation*

The Uniform Cost Search (UCS) algorithm is implemented to find the shortest path between two stations in the transportation graph based on total travel time. UCS is a blind search algorithm that expands the node with the lowest cumulative cost from the start node. The implementation uses a priority queue (heapq) where each element is a tuple containing:

- total_time: the cumulative travel time from the start node
- path: the list of visited stations so far
- moda_path: the sequence of transportation modes taken

```
function UCS(graph, start, goal) → best_path, best_moda, best_time, visited_nodes_count, edges_explored
   queue = [(0, [start], [])] # (time, path, moda)
   best_time : infinite float
   best_path, edges_explored, best_moda : List
   visited : dictionary
   visited_nodes_count = 0

   while queue is not empty do
      total_time, path, moda_path = heapq.heappop(queue)
      current = path[-1]
      visited_nodes_count  = visited_nodes_count + 1

      if current in visited and total_time >= visited[current]:
         continue
      endif
      visited[current] = total_time

      if current = goal then
         if total_time < best_time then
            best_time = total_time
            best_path = path
            best_moda = moda_path
         endif
         continue
      endif

      for neighbor, time_cost, moda in graph[current]['edges']:
         if neighbor not in path:
           penalty = 5 if moda_path and moda_path[-1] != moda else 0
            heapq.heappush(queue, (total_time + time_cost + penalty, path + [neighbor], moda_path + [moda]))
            edges_explored.append((current, neighbor))

   return best_path, best_moda, best_time, visited_nodes_count, edges_explored
```

At each iteration, the node with the least total time is dequeued and processed. If the goal node is reached and the path is optimal—total_time lower time than previously found paths—it is recorded as the best path.

We also calculated a situation if a change in transportation mode is required between the current and next station, a fixed penalty of 5 minutes is added. Otherwise, the heuristic returns 0. This encourages the algorithm to prefer paths that avoid unnecessary transfers between different types of transport. To prevent revisiting nodes unnecessarily, a visited dictionary is maintained to store the lowest time found to reach each station. Additionally, all explored edges are recorded in edges_explored for visualization purposes.

B. *A* Implementation*

The A* (A-star) algorithm extends Uniform Cost Search by incorporating a heuristic function to estimate the remaining cost from the current node to the goal. This heuristic helps prioritize nodes that are likely to lead to a faster solution, making the search more efficient. In this implementation, each element in the priority queue (queue) is a tuple consisting of:

- f: the total estimated cost ($f(n) = g(n) + h(n)$)
- g: the actual cost from the start node to the current node
- path: the list of visited stations
- moda_path: the sequence of transportation modes taken so far

```
function euclidean_distance(node, goal, graph):
        x1, y1 = graph[node]['coords']
        x2, y2 = graph[goal]['coords']
        return math.dist((x1, y1), (x2, y2)
```

```
function astar(graph, start, goal) → best_path, best_moda, best_time, visited_nodes_count, edges_explored
   queue = [(0, [start], [])] # (time, path, moda)
   best_time : infinite float
   best_path, edges_explored, best_moda : List
   visited : dictionary
   visited_nodes_count = 0

   while queue:
      f, g, path, moda_path = heapq.heappop(queue)
      current = path[-1]
      print(f"Current atau path[-1]: {current}")
      visited_nodes_count += 1

      if current in visited and g >= visited[current]:
            print(f"current in visited and total_time >= visited[current]: {g} >= {visited[current]}")
         continue
      visited[current] = g

      if current == goal:
         if g < best_time:
            best_time = g
            best_path = path
            best_moda = moda_path
         continue

      for neighbor, time_cost, moda in graph[current]['edges']:
         if neighbor not in path:
            current_moda = moda_path[-1] if moda_path else None
            penalty = 5 if current_moda and current_moda != moda else 0

            new_g = g + time_cost + penalty
            h = euclidean_distance(neighbor, goal, graph) * 2.5
            f = new_g + h


           heapq.heappush(queue, (f, new_g, path + [neighbor], moda_path + [moda]))
           edges_explored.append((current, neighbor))

   return best_path, best_moda, best_time, visited_nodes_count, edges_explored
```

A key component of A* is the heuristic function. In this project, the heuristic is based on distance between live-node or the neighbor of current node to the goal node or well-known as *eucledian distance*.

The algorithm proceeds similarly to UCS. We calculated a situation if a change in transportation mode is required between the current and next station, a fixed penalty of 5 minutes is added. Otherwise, the heuristic returns 0. This encourages the algorithm to prefer paths that avoid unnecessary transfers between different types of transport. Besides, for *f(n)* A* calculates *f(n) = g(n) + h(n)* for each neighbor node, and nodes are prioritized based on this estimated total cost. The visited dictionary ensures that only the lowest-cost paths to each node are explored, and cycles are avoided by checking whether the neighbor is already in the current path.

C. Heuristics Implementation

1) Eucledian Distance

In the A* algorithm implementation, we uses a heuristic function: Euclidean distance. The purpose of the heuristic is to provide an informed estimate of the remaining cost from a node to the goal, thereby guiding the search more efficiently than uninformed strategies like UCS. To represent spatial proximity between two nodes, each transportation stop is assigned a coordinate in a simplified 2D grid system. The Euclidean distance is calculated using the formula:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}.$$

Fig. 4.1 Lebak Bulus to Bekasi Barat with UCS
(Source: https://en.wikipedia.org/wiki/Euclidean_distance)

This component helps the algorithm prioritize nodes that are geographically closer to the goal. Besides this heuristic, we also combine it with mode-switching penalty so that the route with a situation if a change in transportation mode is required between the current and next station will more likely be skipped.

V. Testing and Analysis

A. Testing

We do experiment on several routes that makes travelers to change transportation mode and the distance between two places is quite far. The goal is to give illustration how UCS and A* algorithms affect route selection since A* using different approach which is eucledian distance as a heuristic.

1) Lebak Bulus to Bekasi Barat

```
Enter START location: Lebak Bulus
Enter GOAL location: Bekasi Barat

Choose search algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
Enter 1 or 2: 1

The fastest route from Lebak Bulus to Bekasi Barat:
Lebak Bulus (MRT) →
Blok M / ASEAN (TJ) →
Pancoran (TJ) →
Cikoko (TJ) →
LRT Cawang (LRT) →
Bekasi Barat
Total travel time: 82 minutes
Nodes visited: 49
Time taken: 0.00000 µs
```

Fig. 5.1 Lebak Bulus to Bekasi Barat with UCS

```
Enter START location: Lebak Bulus
Enter GOAL location: Bekasi Barat

Choose search algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
Enter 1 or 2: 2

The fastest route from Lebak Bulus to Bekasi Barat:
Lebak Bulus (MRT) →
Blok M / ASEAN (TJ) →
Pancoran (TJ) →
Cikoko (TJ) →
LRT Cawang (LRT) →
Bekasi Barat
Total travel time: 82 minutes
Nodes visited: 49
Time taken: 0.00000 µs
```

*Fig. 5.2 Lebak Bulus to Bekasi Barat with A**

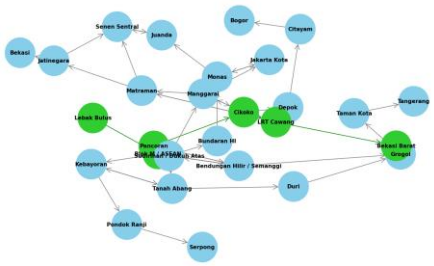

Fig. 5.3 Lebak Bulus to Bekasi Barat with UCS Graph

Fig. 5.4 Lebak Bulus to Bekasi Barat with A Graph*

2) Cikarang to Blok M / ASEAN

```
Choose search algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
Enter 1 or 2: 1

The fastest route from Cikarang to Blok M / ASEAN:
Cikarang (KRL) →
Bekasi (KRL) →
Jatinegara (KRL) →
Matraman (KRL) →
Manggarai (KRL) →
Sudirman / Dukuh Atas (MRT) →
Bendungan Hilir / Semanggi (MRT) →
Blok M / ASEAN
Total travel time: 72 minutes
Nodes visited: 48
Time taken: 0.00000 µs
```

Fig. 5.5 Cikarang to Blok M / ASEAN with UCS

```
Choose search algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
Enter 1 or 2: 2

The fastest route from Cikarang to Blok M / ASEAN:
Cikarang (KRL) →
Bekasi (KRL) →
Jatinegara (KRL) →
Matraman (KRL) →
Manggarai (KRL) →
Sudirman / Dukuh Atas (MRT) →
Bendungan Hilir / Semanggi (MRT) →
Blok M / ASEAN
Total travel time: 72 minutes
Nodes visited: 48
Time taken: 0.00000 µs
```

*Fig. 5.6 Cikarang to Blok M / ASEAN with A**

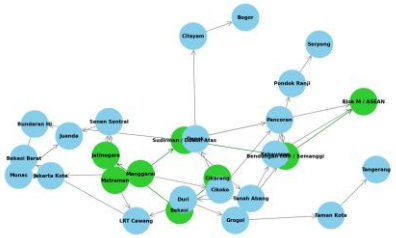

Fig. 5.7 Cikarang to Blok M / ASEAN with UCS Graph

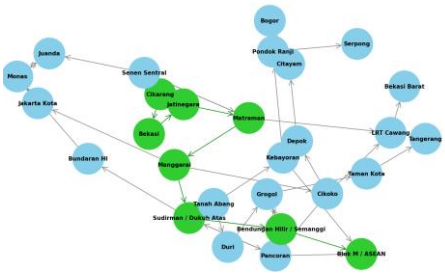

Fig. 5.8 Cikarang to Blok M / ASEAN with A Graph*

3) *Serpong to Monas*

```
Enter START location: Lebak Bulus
Enter GOAL location: Bekasi Barat

Choose search algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
Enter 1 or 2: 1

The fastest route from Lebak Bulus to Bekasi Barat:
Lebak Bulus (MRT) →
Blok M / ASEAN (TJ) →
Pancoran (TJ) →
Cikoko (TJ) →
LRT Cawang (LRT) →
Bekasi Barat
Total travel time: 82 minutes
Nodes visited: 49
Time taken: 0.00000 µs
```

Fig. 5.9 Serpong to Monas with UCS

```
Enter START location: Lebak Bulus
Enter GOAL location: Bekasi Barat

Choose search algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
Enter 1 or 2: 2

The fastest route from Lebak Bulus to Bekasi Barat:
Lebak Bulus (MRT) →
Blok M / ASEAN (TJ) →
Pancoran (TJ) →
Cikoko (TJ) →
LRT Cawang (LRT) →
Bekasi Barat
Total travel time: 82 minutes
Nodes visited: 49
Time taken: 0.00000 µs
```

*Fig. 5.10 Serpong to Monas with A**

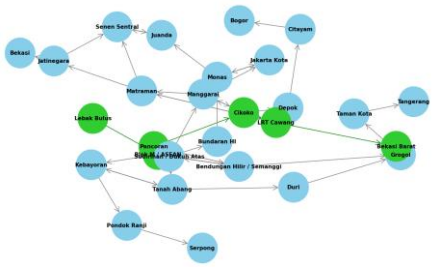

Fig. 5.11 Serpong to Monas with UCS Graph

Fig. 5.12 Serpong to Monas with A Graph*

B. *Analysis*

From the previous testing part, several routes are being used as a testcase. We deliberately selected these routes according to their distance and how complicated it would be if a traveler gets through it. The goal node of these routes cannot be accessed directly by one or even two modes of transportation. Nevertheless, all the things we have done seem to be quite useless.

In testing, three routes with UCS and A* algorithm respectively showing exactly same result. From *1) Lebak Bulus to Bekasi Barat*, although traveler needs to change mode of transportation two times and the travel time is at its best, both algorithms give with identical result including the path and its transportation mode, total travel time, visited nodes and time taken to process this problem. Unexpectedly, the other two testing give the same pattern and result. All of this could be possible due to several reasons or hypothesis.

1) *Small Graph Scale*

In this research, we only use about ~40 stations or bus stops in order to simplify the actual large scale of Jabodetabek's public transportation network. Then, the best or optimal path could be found without many explorations. Its consequence is both UCS and A* algorithm will likely explore the same route.

2) *Small Value of Euclidean Distance*

The value of *g(n)* is usually on range between 70-100 minutes whereas the value of *h(n)* (Euclidean times 1.5) is only about 5-10. This quite large gap makes A* algorithm is not 'directed' enough and similar to UCS

3) *Dominance of a Single Optimal Route*

In many of the tested routes, the structure of the transportation network strongly favors a single optimal path. This means that both Uniform Cost Search (UCS) and A* are essentially "forced" to follow the same route regardless of heuristic guidance, simply because alternative routes are either significantly longer or require more mode switches. As a result, even though A* uses heuristic to prioritize certain paths, it gives the same solution as UCS due to the natural dominance of that route within the graph.

4) *Mode Switching Penalty Has Limited Influence*

Although a fixed penalty is applied when switching transportation modes, its magnitude (e.g., 5 minutes) is often not large enough to significantly affect the selection of routes. Most available paths between distant nodes already involve one or more mode switches, making the penalty a consistent factor across alternatives. Consequently, it does not meaningfully differentiate between paths and has little impact on the decision-making process of either algorithm.

VI. Conclusion

This study aimed to explore the application of graph search algorithms, particularly Uniform Cost Search (UCS) and A* search, for optimizing the fastest public transportation routes within the Jakarta Metropolitan Area. By constructing a simplified transportation network graph using real-world data from KRL, MRT, LRT, and Trans Jakarta routes, we evaluated the effectiveness of both algorithms in terms of travel time, number of visited nodes, and overall performance.

The A* algorithm used a heuristic function based on Euclidean distance between transit points, along with a penalty for switching transportation modes. However, the experimental results showed that both UCS and A* consistently produced the same optimal routes and travel times across all test cases. In many instances, the number of nodes visited by A* was also comparable to or greater than that of UCS.

These findings suggest that in small-scale, structured, and dominantly linear graphs like the one used in this study, the advantage of heuristic-based search is diminished. The network's limited complexity and the presence of strongly favored optimal routes made both algorithms showing to the same solutions.

Nonetheless, this work validates the applicability of both UCS and A* for route optimization problems and highlights the critical role of heuristic design and network complexity in determining algorithmic efficiency.

Future work may involve extending the transportation graph with higher node density, incorporating real-time traffic data, or evaluating alternative heuristic functions to better exploit the advantages of informed search strategies.

GITHUB AND DATA LINK

Github: https://github.com/hnfadtya/MakalahStima_13523041

DataSheet:https://docs.google.com/spreadsheets/d/13nZYXq2VpAIx09Ud2yISfSEN5PEKpsuK0588862Fle8/edit?gid=0#gid=0

ACKNOWLEDGMENT

Alhamdulillah and all praise to Allah for His mercy, we are able to finish "Optimization of Fastest Public Transportation Route Selection in Jakarta Metropolitan Area Using Uniform Cost Search and A* Algorithm" paper. Also, my gratitude sent to my lecturer, Dr. Nur Ulfa Mauladevi, S.T., M.Sc., for her guidance and patience to provided beneficial and precious knowledge along this semester. Thank you.

REFERENCES

[1] Munir, Rinaldi. 2025. BFS, DFS, UCS, Greedy Best First Search. [online] Available at: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf> [Accessed 23 June 2025 14:56]

[2] Munir, Rinaldi. 2025. BFS, DFS, UCS, Greedy Best First Search. [online] Available at: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-(2025)-Bagian2.pdf> [Accessed 23 June 2025 14:56]

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Juni 2025

Hanif Kalyana Aditya/13523041