Efficient Route Mapping in the Singapore MRT Leveraging Graph Theory and Dijkstra's Algorithm

Filbert - 13522021[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522021@std.stei.itb.ac.id

***Abstract*—With the ever-growing urban sprawl, efficient public transportation systems are crucial for the sustainability of metropolitan areas. Singapore's Mass Rapid Transit (MRT) stands as a paragon of such systems, offering swift transit across the city-state. This paper presents a novel approach to route optimization within the MRT network by harnessing the computational prowess of graph theory and Dijkstra's algorithm. Our study delves into the intricacies of the MRT's infrastructure, translating it into a weighted graph that encapsulates the myriad of routes, transit times, and interchange complexities. By implementing Dijkstra's algorithm, we compute the most expedient paths between any two given stations, factoring in the dynamic nature of peak and off-peak travel times. The results demonstrate a significant enhancement in route planning efficiency, providing a robust tool that commuters and system planners can utilize for improved journey planning and network management**

***Keywords*—Singapore MRT, Graph, Dijkstra, Effective route.**

I. INTRODUCTION

As urban populations swell and the demand for timely and efficient transportation escalates, the role of rapid transit systems becomes increasingly cardinal. In the densely woven urban tapestry of Singapore, the MRT is not merely a convenience but a lifeline that threads through the city, binding it together. This paper explores the application of graph theory and Dijkstra's algorithm to distill an optimized mapping solution for navigating the complex MRT network.

Graph theory provides a natural framework for modeling the interconnected stations and transit lines, while Dijkstra's algorithm serves as a beacon, guiding commuters through the shortest paths amidst the labyrinthine network. The crux of this study lies in its adaptation of these classical theories to accommodate the unique operational characteristics of the MRT system, such as varying travel times during peak and off-peak hours and the strategic placement of interchange stations.

The objective of this paper is twofold: firstly, to construct a comprehensive graph representation of the MRT network that captures the nuances of its topology; and secondly, to implement a refined version of Dijkstra's algorithm that accounts for temporal fluctuations in travel time, thereby delivering a pragmatic and efficient route mapping methodology. Through this dual-pronged approach, the paper aims to contribute to the domain of urban transit planning and to proffer a methodological blueprint for similar applications in metropolitan transit systems worldwide.

II. BASIC THEORY

A. Graph Definiton

A graph is a mathematical construct used to depict the relationships between discrete elements. It comprises vertices, which can be either linked or unlinked, typically denoted as dots or circles on the graph, and edges, which are connections between pairs of vertices represented as lines connecting these dots or circles.

In a formal representation, a graph is denoted as $G = (V, E)$, where V is a non-empty collection of vertices, and E is a set of edges. Each edge connects one or two vertices, known as its endpoints, establishing a connection between these vertices.

B. Graph Types

Based on the presence and absence of loops and multiple edges connecting the same vertices, the graph has two types:

1. Simple graphs

   A simple graph is characterized by the following properties: it only contains edges that connect two distinct vertices, and there are no duplicate edges connecting the same pair of vertices. In a simple graph, each edge is linked to an unordered pair of vertices, and no other edge is associated with this particular pair of vertices.

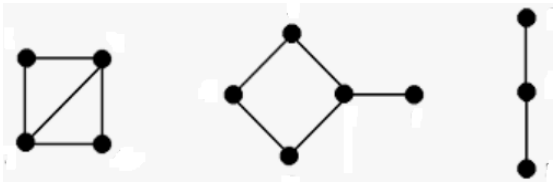

Fig. 1. Simple Graphs (Source: [1])

2. Unsimple graph

   Unsimple graphs are those in which loops are present or multiple edges connect the same vertices. Unsimple graphs can be further divided into two subcategories:

   a. Multipgraphs

      Multipgraphs are graphs that have multiple edges connecting the same vertices. When there are m different edges associated with the same unordered pair of vertices $\{u, v\}$, then $\{u, v\}$ is an edge of multiplicity m. This feature is particularly useful in modeling scenarios where different types of connections or relationships between the same entities exist.

b. Pseudograph

Pseudographs are graphs allow for both loops (edges connecting a vertex to itself) and multiple edges. They are useful in scenarios where self-connections are meaningful or when multiple relationship between the same entities need representation.

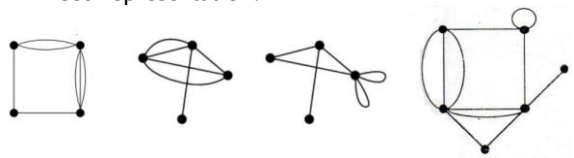

Fig. 2. Unsimple Graphs (Source: [1])

Based on the direction of the edges, graphs can be divided into two types:

1. Undirected graph

In an undirected graph, each edge does not have any direction. They're used when the direction of a relationship is not a concern, such as in social network friendships.

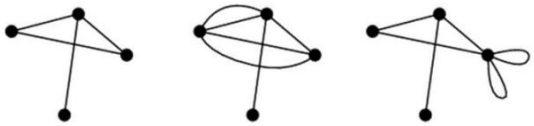

Fig. 3. Undirected Graphs (Source: [1])

2. Directed Graph

A directed graph (V, E) consists of a nonempty set of vertices V and a set of directed edges E. Each directed edge is associated with an ordered pair of vertices. Edge with the ordered pair $\{u, v\}$ is an edge that starts at u and ends at v. This structure is essential in scenarios like web page links or city road maps, where direction matters.

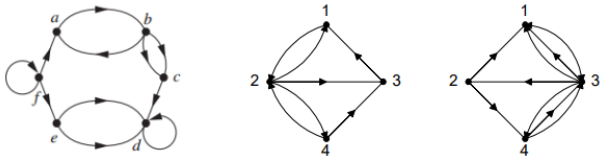

Fig. 4. Directed Graphs (Source: [1])

There are also some type of special graph:

1. Complete Graph

These are graphs where every vertex is connected to every other vertex with a unique edge. They represent the most interconnected scenario possible in a graph.

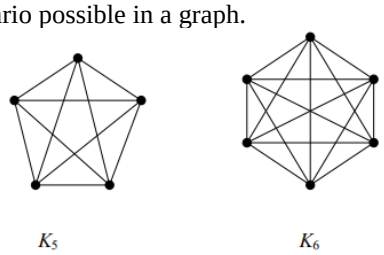

Fig. 5. Complete Graphs (Source: [1])

2. Circle Graph

These graphs form a continuous loop, with each vertex connected to exactly two others, forming a circular structure. They are a special case of regular graphs.

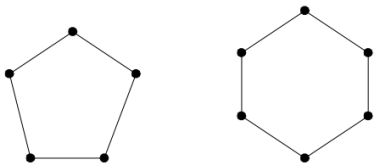

Fig. 6. Circle Graphs (Source: [1])

3. Regular Graph

A graph where each vertex has the same degree, meaning the same number of edges connected to each vertex. This uniformity can be crucial in network design and symmetry analysis.

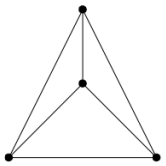

Fig. 7. Regular Graphs(Source: [1])

4. Bipartite Graph

These can be split into two distinct sets of vertices with edges only running between vertices of different sets. They are particularly useful in scenarios like job assignments or matching problems, where two distinct types of entities are involved.

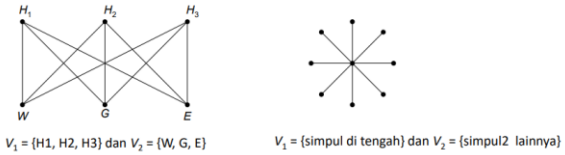

Fig. 8. Bipartite Graphs (Source: [1])

C. Graph Terninologies

In graph theory, there are several terms used when analyzing graphs. The terminologies that will be used are as follows:

1. Adjacent

In an undirected graph, two vertices are considered adjacent if they are connected by an edge. In a graph with directed edges, the vertex at the beginning of the edge is considered adjacent to the vertex at the end of the edge. When (u, v) is an edge of the graph G, u is the initial vertex of an and v is the terminal or end vertex. If an edge forms a loop, the initial and terminal vertices are the same.

2. Incidence

An edge (u, v) that connects u and v is called an incident with the vertices u and v.

3. Degree

In an undirected graph, the degree of a vertex is defined as the number of edges that are incident with it. If a loop is present at a vertex, it is counted twice in the degree of that vertex. The degree of a vertex is represented by the notation $deg(v)$. A vertex

with no incident edges is referred to as an isolated vertex. When the entire graph consists solely of isolated vertices, it is called a null or empty graph.

4. Path

   A path is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along the edges of the graph. As the path progresses, it passes through the vertices that are the endpoints of these edges. Two vertices are considered connected if there is a path between them. A graph is said to be connected if every pair of vertices is connected. In the case of a directed graph, strong connectivity refers to a directed Makalah IF2120 Matematika Diskrit – Sem. I Tahun 2023/2024 path from u to v and a directed path from v to u for every pair of vertices u, v. Weak connectivity, on the other hand, refers to a directed graph that produces a connected graph only if all of its directed edges are replaced with undirected ones.

5. Cycle

   A cycle is a sequence of vertices and edges that starts and ends at the same vertex. In other words, it is a path that does not have a specific starting or ending point but instead forms a closed loop. The length of a cycle is the number of edges it contains, and a graph can have multiple cycles of varying lengths.

6. Subgraph

   A subgraph is a subset of the vertices and all the edges of a larger graph. This subset forms a smaller graph that maintains the same properties and connections as the original graph.

7. Weighted Graph

   A weighted graph is a graph that assigns weights, or numerical values, to each of its edges. These weights can represent a variety of things, such as distance, price, cost, or queue. A weighted graph can be implemented in both directed and undirected graphs.

8. Isolated Vertex

   Isolated Vertex means a vertex with no connections (edges) to other vertices. It's significant in identifying outliers or standalone nodes in a network.

9. Null Graph or Empty Graph

   Null Graph is a graph without any edges. It's a theoretical concept, representing the most disconnected scenario.

10. Cut set

    Cut set is a group of edges whose removal increases the graph's disconnected components. It's a critical concept in network reliability and vulnerability analysis.

11. Connected

    A graph is said to be connected if there is a path between every pair of vertices. This property is crucial in ensuring network coherence and accessibility.

D. Dijkstra

Dijkstra's algorithm is a well-established computational procedure for finding the shortest paths between nodes in a graph, which can be either weighted or unweighted. It is particularly efficient for graphs with non-negative edge paths, making it ideal for applications in routing and navigation systems where the shortest or quickest route is desired. The algorithm operates by iteratively selecting the nearest unvisited vertex, calculating the distance through it to each of its neighboring vertices, and updating the path if it yields a shorter distance. For the Singapore MRT, the algorithm's implementation considers the transit time between stations, with peak and off-peak hours influencing the weights of the edges.

Enhancing its application, the study employs Dijkstra's algorithm to traverse the graph model of the MRT, seeking the time-efficient route for commuters. A notable augmentation in this study is the algorithm's dynamic adjustment of edge weights, mirroring real-time variations in peak and off-peak travel times. This adaptability injects a layer of realism into the model, acknowledging and addressing the temporal dynamics that play a crucial role in influencing commuter decisions and route preferences. By integrating these temporal factors, the algorithm transcends its traditional application, offering a more nuanced and realistic navigation solution for complex urban transit systems.

III. METHODOLOGY

A. Limitations

When it comes to analyzing the most optimized route within the Singapore MRT system, several challenges and limitations must be considered to make the calculation more manageable and less complex. These limitations are essential for understanding the methodology employed in this study:

1. Dummy Scale for Distance/Time

   To represent the travel distance or time between MRT stations, the author employs a simplified scale ranging from 1 to 5. This scale is used as a proxy for actual time or distance values, which can vary considerably in real-world scenarios. The use of this dummy scale allows for a more straightforward mathematical model but may not capture the precise nuances of travel times in the MRT system.

2. Static Data

   It's important to note that all the data used in this analysis is static and does not consider dynamic changes that occur in the real Singapore MRT system. In reality, MRT schedules, station conditions, and travel times can be subject to change due to factors such as maintenance, construction, or unforeseen events. Therefore, the analysis is based on a snapshot of the MRT system at a specific point in time and does not account for potential alterations in the future.

B. Location Data

Singapore is a relatively small country with a well-developed transportation network. Currently, it has more than 140 MRT stations. This can be both confusing and convenient for people when it comes to

getting to their destinations quickly or even potentially taking longer routes because they are not using the shortest or most optimal distances, which can be disadvantageous in terms of both time and money.

To facilitate the calculation of the most optimal and shortest routes within the Singapore MRT network, it is crucial to represent MRT stations as nodes in a graph. This representation serves as the foundation for applying algorithms like Dijkstra's to find the best paths. Here's a closer look at how the author accomplishes this in the paper:

Fig. 9. Singapore MRT map (Source: Primary)

In this paper, the author compiles a comprehensive list of all MRT stations based on data obtained from the official Singapore MRT map. This list includes the names of each station and their unique identifying information. To construct the graph representation, each MRT station is connected to every other station within the network using edges. These edges signify the possible routes between stations and are essential for route optimization. The author determines the "distance" or "cost" associated with each edge, utilizing the previously mentioned 1-5 scale. These values represent the perceived ease or difficulty of traveling between stations and guide the route-finding algorithm.

```
blue_stations = ["Bukit Panjang","Cashew", "Hillview", "Beauty World", "King Albert Park", "Sixth Avenue",
 "Tan Kah Kee", "Botanic Gardens", "Stevens", "Newton", "Little India", "Rochor", "Bugis",
 "Promenade", "Bayfront", "Downtown", "Telok Ayer", "Chinatown", "Fort Canning", "Bencoolen",
 "Jalan Besar", "Bendemeer", "Geylang Bahru", "Mattar", "MacPherson", "Ubi", "Kaki Bukit",
 "Bedok North", "Bedok Reservoir", "Tampines West", "Tampines", "Tampines East", "Upper Changi", "Expo"]

blue_times= [2,2,3,2,2,2,2,2,2,3,1,2,2,2,2,1,2,2,2,1,2,2,2,2,2,2,2,2,3,2,2,3,2]
```

Fig. 10. Stations Data used in Program (Source: Primary)

All station names and their corresponding edge weights (on the 1-5 scale) are organized and stored in a data structure, typically an array or a similar data container. This structured data provides the algorithm with the necessary information to navigate the network efficiently.

```
interchanges = {
    "Chinatown":["NE3", "DT19"], "Little India": ["NE6", "DT11"], "Bugis" :
    ["DT13", "EW12"], "Outram Park" : ["NE2", "EW16", "TE16"], "Tampines":
    ["DT31", "EW2"], "Jurong East": ["NS1", "EW24"], "Newton": ["NS20",
    "DT10"], "Dhoby Ghaut": ["NS23", "NE5", "CC1"], "City Hall": ["NS24",
    "EW13"], "Raffles Place": ["NS25", "EW14"],"Promenade": ["CC4", "DT14"],
    "Bayfront":["CE1", "DT15"], "Marina Bay": ["NS26","TE19"], "Paya Lebar":
    ["CC9", "EW8"], "MacPherson": ["CC10", "DT25"], "Serangoon":["NE11",
    "CC13"], "Bishan":["NS16", "CC15"], "Caldecott":["CC17","TE9"], "Botanic
    Gardens":["CC19", "DT8"], "Buona Vista":["CC22", "EW21"],"Harbour Front":
    ["CC29", "NE1"],"Woodlands":["NS8","TE2"],"Stevens":["TE10","DT9"],
    "Orchard":["TE13","NS21"]
}
```

Fig. 11. Interchanges Stations data (Source: Primary)

Additionally, the author identifies and maintains a list of interchange stations. Interchanges are crucial points within the MRT network where passengers can switch between different lines or routes. These interchange stations are saved in a set, which includes both station names and station codes, ensuring that the algorithm can identify and optimize routes through these key hubs.

C. Dijkstra Algorithm

The Dijkstra's algorithm is employed here to find the shortest path between two points in a weighted graph. It operates on a graph represented as an adjacency list, where each node is connected to other nodes by edges with associated weights. In my dijsktar algorithm, it accepts 3 parameters, graph, start which is the starting mrt stations code and end is the end of mrt stations code. The algorithm starts with initializing data structures to keep track of the shortest path information.

```
def dijkstra(graph, start, end):
    shortest_paths = {start: (None, 0)}
    visited = set()
    current_node = start

    while current_node != end:
        visited.add(current_node)
        destinations = graph[current_node]
        weight_to_current = shortest_paths[current_node][1]

        for next_node, weight in destinations.items():
            new_weight = weight + weight_to_current
            if new_weight < shortest_paths.get(next_node, (None, float('inf')))[1]:
                shortest_paths[next_node] = (current_node, new_weight)

        next_destinations = {node: shortest_paths[node] for node in shortest_paths if node not in visited}
        if not next_destinations:
            return "Route Not Possible"

        current_node = min(next_destinations, key=lambda k: next_destinations[k][1])

    path = []
    while current_node:
        path.append(current_node)
        current_node = shortest_paths[current_node][0]

    return path[::-1]
```

Fig. 12. Djikstra Algortihm Implementation (Source: Primary)

It begins at a designated starting point and sets its distance to Zero. Visited is used to keep track of visited nodes to avoid revisiting them. Current node variable is set to starting node to initiate the traversal.

The loop will continues until current_node reaches the end of the destination. Within each iteration, the algorithm marks the "current_node" as visited, retrieves neighboring nodes and their associated edge weights from the graph, calculates new distances through the "current_node," and updates the shortest path if a shorter route is identified. Unvisited neighboring nodes and their corresponding shortest paths are stored in the "next_destinations" dictionary. If no unvisited nodes remain in "next_destinations," signifying an absence of a valid path from the start to the destination, the algorithm returns "Route Not Possible." Otherwise, it proceeds by selecting the node with the shortest distance among the unvisited nodes as the new "current_node." This process repeats until the destination is reached or until it is determined that no valid route exists.

After reaching the ending node, create an empty list called "path". We need to trace back from the ending node to the starting node using the "shortest_paths" dictionary, appending each node to the "path". Reverse the "path" list to obtain the correct order of nodes from start to end and return the shortest path as a list.

This is the example of the list of the mrt codes process from Punggol to Orchard :

```
List Shortest Path :  ['NE16', 'NE15', 'NE14', 'NE13', 'NE12', 'NE11', 'NE10', 'NE9',
'NE8', 'NE7', 'NE6', 'NE5', 'NS23', 'NS22', 'NS21']
```

Fig. 13. Output Result (Source: Primary)

D. Most Optimal Route Analysis

In the theoretical framework and problem modeling conducted by the author, the optimized Singapore MRT route was achieved using the Python programming language. Implementation began with sourcing data from Wikipedia, which provides a list of the newest MRT stations. This data was manually input into the code editor, as no API was used for data retrieval. Upon determining the shortest path, users have the option to visualize the graph—though it should be noted that the graph layout may not reflect the actual physical layout of the Singapore MRT stations.

Before proceeding further, the author will outline the program flow. Initially, all station data, stored in a list, generates respective station codes that correlate with the actual MRT identifiers. Subsequently, the graph is constructed, albeit utilizing only MRT station codes. The main function then call Dijkstra's algorithm to calculate the shortest path and presents the output with both station names and codes for an improved user interface.

```python
def generate_station_codes(stations, line_prefix):
    station_codes = {}
    for i, station in enumerate(stations):
        code = f"{line_prefix}{len(stations) - i}"
        station_codes[station] = code
    return station_codes

def generate_station_codes_from_start(stations, line_prefix):
    station_codes = {}
    for i, station in enumerate(stations):
        code = f"{line_prefix}{i + 1}"
        station_codes[station] = code
    return station_codes

green_station_codes = generate_station_codes(green_stations, "EW")
blue_station_codes =  generate_station_codes_from_start(blue_stations, "DT")
purple_station_codes =  generate_station_codes(purple_stations, "NE")
red_station_codes =  generate_station_codes_from_start(red_stations, "NS")
circle_station_codes = generate_station_codes_from_start(circle_stations, "CC")
thomson_station_codes = generate_station_codes_from_start(thomson_stations, "TE")
```

Fig. 14. Generate codes for each Stations (Source: Primary)

```python
def build_graph(stations, times, graph, station_codes, interchange_codes):
    for i in range(len(stations) - 1):
        station1 = stations[i]
        station2 = stations[i + 1]
        time = times[i]

        station1_code = station_codes[station1]
        station2_code = station_codes[station2]

        if station1_code not in graph:
            graph[station1_code] = {}
        if station2_code not in graph:
            graph[station2_code] = {}

        graph[station1_code][station2_code] = time
        graph[station2_code][station1_code] = time

    # Handle interchange stations
    for station in interchange_codes:
        codes = interchange_codes[station]
        for i in range(len(codes)):
            for j in range(i + 1, len(codes)):
                if codes[i] not in graph:
                    graph[codes[i]] = {}
                if codes[j] not in graph:
                    graph[codes[j]] = {}
                graph[codes[i]][codes[j]] = 3  # weight for transfer lines
                graph[codes[j]][codes[i]] = 3
```

Fig. 15. Build Graph Implementation (Source: Primary)

```python
graph = {}

build_graph(blue_stations, blue_times, graph,blue_station_codes,interchanges)
build_graph(purple_stations, purple_times, graph,purple_station_codes,interchanges)
build_graph(green_stations, green_times, graph, green_station_codes,interchanges)
build_graph(red_stations, red_times, graph,red_station_codes,interchanges)
build_graph(circle_stations, circle_times, graph,circle_station_codes,interchanges)
build_graph(thomson_stations, thomson_times, graph,thomson_station_codes,interchanges)
```

Fig. 16. Create Graph (Source: Primary)

For the first case study, it was conducted with the source station set as Bukit Panjang and the destination as Tampines West. Although a direct route exists, spanning 29 stops on the same line, the program is designed to identify a more efficient path. Remarkably, the suggested route reduces the journey by nearly ten stops, albeit with two line changes. This optimization exemplifies the program's capability to enhance travel efficiency. Below are the details of the identified optimal route, alongside an option for graphical visualization:

```
Welcome to the Singapore MRT Route Finder!
Let's find the most optimized route for your journey.
Enjoy your trip planning with us! ^.^

Do you want to see the list of the MRT stations (YES/NO): no

Enter the source MRT station: bukit panjang
Enter the destination MRT station: tampines west

The shortest path is : Bukit Panjang (DT1) -> Cashew (DT2) -> Hillview (DT3) -> Beauty World (DT4) -> King
Albert Park (DT5) -> Sixth Avenue (DT6) -> Tan Kah Kee (DT7) -> Botanic Gardens (DT8) -> Changing Stations
at Botanic Gardens from DT line to CC line -> Botanic Gardens (CC19) -> Caldecott (CC17) -> Marymount (CC16
) -> Bishan (CC15) -> Lorong Chuan (CC14) -> Serangoon (CC13) -> Bartley (CC12) -> Tai Seng (CC11) -> MacPh
erson (CC10) -> Changing Stations at MacPherson from CC line to DT line -> MacPherson (DT25) -> Ubi (DT26)
-> Kaki Bukit (DT27) -> Bedok North (DT28) -> Bedok Reservoir (DT29) -> Tampines West (DT30)
Total travel time : 55 minutes

Thank you for using the Singapore MRT Route Finder. Have a great trip!

Last but not least, Do you want to view the visualization of the graph (YES/NO) ? yes

Goodbye!!!
```

Fig. 17. First Study Case (Source: Primary)

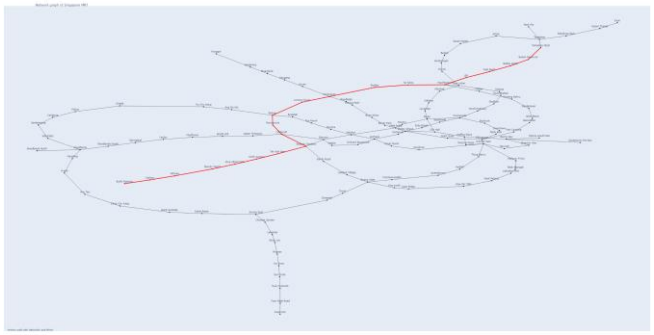

Fig. 18. Graph visualtion (Source : Primary)

The network graph, composed using NetworkX and Plotly, delineates all connections between nodes with black lines, while the computed shortest path is highlighted in red. This visualization is not a replica of the actual Singapore MRT layout, as such fidelity would require precise geographic coordinates or API access to station location data. Nonetheless, the graphical representation is instrumental in illustrating the algorithm's function and the route's practical application.

In this program, the total time consumed from Bukit Panjang to Tampines West is roughly 55 minutes. However, the trade-off is that the user will later need to change lines twice, which some people may not prefer to do. If the user chooses to take only one line, it will take about 71 minutes to reach Tampines West, which is 16 minutes slower. Dijkstra's algorithm prioritizes the cumulative "weight" of the journey, where the weight is a measure of time rather than distance. This weight takes into account various factors, including the number of stops and interchange penalties, to calculate the most time-effective route. Consequently, the algorithm may suggest a route with more interchanges if it results in a lower overall travel time.

The times the author considers are as follows: for example, from Bukit Panjang to Cashew, it is set to only 2 minutes, while from Sixth Avenue to Tan Kah Kee, it is set to 3 minutes. These times increment as the routes taken from one station to another become longer.

For the second case study, it was conducted with the source station set as Jurong East and the destination as Esplanade. There are two ways to go. The longest route might be from Jurong East, where users transfer only once to the Buona Vista CC line and then directly proceed to Esplanade. However, of course, this route takes significantly more time compared to the other option which is 55 minutes and is slower 17 minutes than the fastest route.

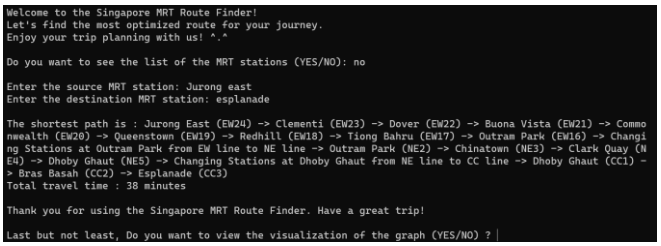

Fig. 19. Second Study Case (Source: Primary)

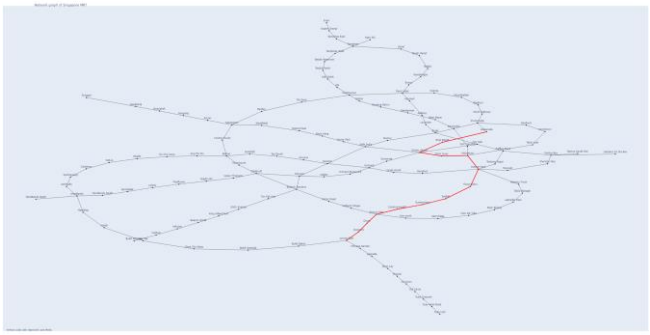

Fig. 20. Graph visualtion (Source : Primary)

The shortest route involves transferring to Outram Park, changing from EW to NE line and then, after only a few stops, users need to transfer lines again at Dhoby Ghaut CC line, finally arriving at Esplanade in just 3 stops. This may be inefficient for users who prioritize convenience within the MRT system and want to minimize line transfers. However, to find the most optimized route, there is always a trade-off. Even though it requires changing lines multiple times, as long as it is the shortest and most optimal route in terms of time, the algorithm will always select it because it offers the most efficient travel time.

For the final case study, the route from Pasir Ris to Tuas Link was assessed. Contrary to the previous cases, this journey is optimally served by a direct line, making it an apparent exception within the context of the algorithm's complexity. The Singapore MRT map indicates that a straightforward route exists, and the program corroborates this by not suggesting any transfers, which would inevitably increase travel time.

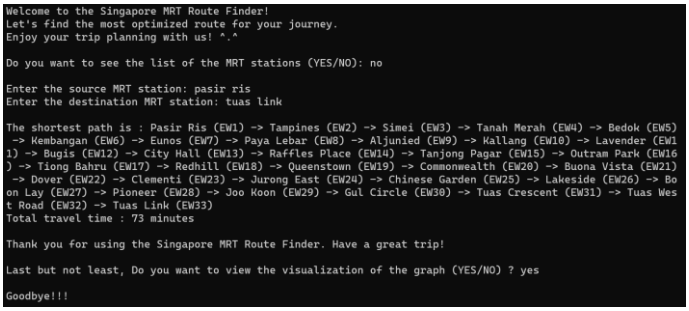

Fig. 21. Third Study Case (Source: Primary)

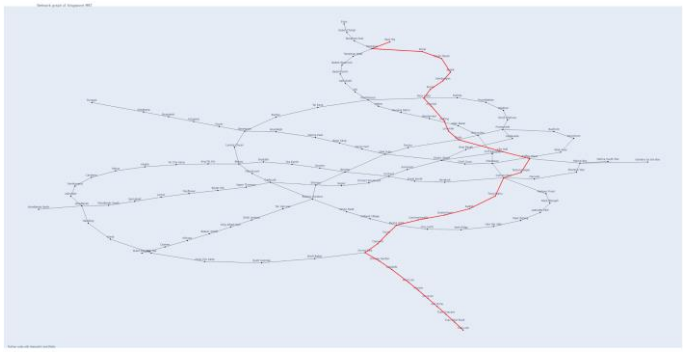

Fig. 22. Graph visualtion (Source : Primary)

The potential alternative involving transfers at Paya Lebar and Buona Vista, despite initially seeming like a viable option, is ultimately inefficient. The program's calculations confirm that the direct route is superior, with transfers adding unnecessary complexity and time due to waiting periods at interchange stations.

This scenario underscores the algorithm's capability to discern not just the shortest path by distance, but the most time-effective route. It affirms the program's utility in offering not just any route, but the most pragmatic option for the commuter. Even in seemingly straightforward scenarios, the algorithm validates the expected human judgment, reinforcing its reliability.

In summarizing the results of the three case studies, we observe the intricate balance between the number of stops, the

necessity of line transfers, and the overall travel time — a balance that is adeptly managed by the application of Dijkstra's algorithm within the Singapore MRT context.

Case Study 1 (Bukit Panjang to Tampines West): Here, the algorithm proved its efficacy by proposing a route that significantly reduced the number of stops when compared to the direct line, showcasing its strategic prowess in optimizing travel routes. The computed path, although involving line changes, exemplified a significant reduction in travel time, affirming the algorithm's superiority over conventional route planning.

Case Study 2 (Jurong East to Esplanade): This scenario highlighted the algorithm's nuanced approach, as it negotiated multiple transfers to curtail the journey's duration. Despite the apparent inconvenience of line changes, the suggested route was validated as the most time-efficient, demonstrating the algorithm's capacity to leverage the complexity of the MRT network to the commuter's advantage.

Case Study 3 (Pasir Ris to Tuas Link): Contrasting with the previous examples, this case affirmed the algorithm's discernment in endorsing a direct route without any transfers, as it was unambiguously the most expedient option. This outcome reinforced the algorithm's versatility — its ability to not only recognize when to minimize stops but also when to endorse a straightforward path.

Across all cases, the algorithm consistently delivered routes that either saved time or confirmed the expected optimal path, thereby showcasing its effectiveness as a route optimization tool. It navigated the intricacies of the MRT network with precision, substantiating the benefit of algorithmic assistance in urban transit systems.

The results of these case studies contribute to a broader understanding of how computational algorithms can be applied to enhance the efficiency of public transportation systems. By meticulously calculating the quickest routes and presenting them in a user-friendly manner, the program not only serves as a testament to the practical applications of graph theory and Dijkstra's algorithm but also sets a precedent for future innovations in the field of transit planning and management.

IV. CONCLUSION

This study successfully demonstrated the application of graph theory and Dijkstra's algorithm in optimizing MRT routes in Singapore. The developed program transcends the limitations of manual route planning by considering a multitude of factors such as interchange stations, peak and off-peak hours, and wait times. The case studies presented within this paper illustrate the program's efficacy in providing the most time-efficient travel routes, a testament to the power of algorithmic solutions in urban transportation planning.

The program could be further enhanced by integrating real-time data, which would allow it to adapt to dynamic conditions such as service disruptions, track maintenance, or even real-time traffic conditions. Such advancements could transform the program from a static route optimizer into a dynamic travel assistant, catering to the evolving needs of a modern metropolis's inhabitants.

In conclusion, the exploration of the Singapore MRT system through this program has provided valuable insights into the practical application of theoretical computational concepts. It has set a foundation for further research and development in the field of urban transportation efficiency, potentially impacting millions of commuters by saving time and improving travel experiences.

V. APPENDIX

The complete program of this paper can be found below.
https://github.com/Filbert88/Graph-and-Dijkstra-SingaporeMRTLine

VI. ACKNOWLEDGMENT

First and foremost, the author expresses gratitude to their Discrete Mathematics lecturer, Dr. Nur Ulfa Maulidevi, S.T., M.Sc., for generously sharing knowledge with fellow students, including the researcher, whose successful completion of this research owes much to their guidance. Additionally, the researcher extends thanks to the other lecturers who contributed to and were part of the IF2120 Discrete Math class, as their dedication played a crucial role in motivating the researcher to write this research paper. Lastly, the researcher acknowledges their friends for their unwavering support throughout the Discrete Math class, fostering a collaborative learning environment that has contributed to personal growth and development over time.

REFERENCES

[1] R. Munir, "Graf Bagian 1," IF2120 Matematika Diskrit. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/Graf-Bagian1-2023.pdf. [Accessed Dec. 8, 2023]

[2] K. H. Rosen, Discrete Mathematics and Its Applications Seventh Edition. New York, America: McGraw-Hill, 2017.

[3] "LTA | Land Transport Authority," lta.gov.sg. [Online]. Available: https://www.lta.gov.sg/content/ltagov/en/getting_around/public_transport/rail_network.html/. [Accessed: Dec. 9, 2023].

[4] Navone, Estefania Cassingena. 2020. [Online]. Available: https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithmvisual-introduction/. [Accessed: Dec 9, 2023].

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, 11 Desember 2023

Filbert 13522021