



# OptiTour: Tourist Transit Optimizer

Choon Keat Ling, Han Xiang Kee, Kevin Ong Jia Ming, Siti Halilah Binte Rosley<sup>(✉)</sup>,  
Zachary Ding Fang Liang, and Huaqun Guo

Singapore Institute of Technology, 10 Dover Dr 138683, Singapore, Singapore  
{2201300, 2201423, 2201380, 2201274,  
2201365}@sit.singaporetech.edu.sg,  
huaqun.guo@singaporetech.edu.sg

**Abstract.** The project aims to develop an optimized travel route system for a tourist travel bus service called “Tourist Transit Optimizer” or “OptiTour”. The OptiTour is designed to find the most cost-effective route for transporting tourists from Changi Airport Terminal 3 to their respective hotels in Singapore. The decision to develop this system was driven by the need to minimize operational costs and maximize efficiency in the highly competitive travel and tourism industry. The pre-existing knowledge base for this project includes understanding the local area, knowledge of the main hotels served, and basics of operations management and optimization algorithms.

**Keywords:** Route optimisation · Travelling Salesman Problem · Shortest path finding

## 1 Introduction

In this paper, we present “OptiTour”, which uses data structures and algorithms to optimize tourist bus routes in Singapore, considering various factors such as road network, Electronic Road Pricing (ERP) rates, and the geographical coordinates of tourist attractions.

Our solution leverages the Dijkstra’s algorithm, a well-known graph traversal algorithm, to find the most cost-effective routes for a tourist bus service. By integrating data from multiple sources and employing a custom implementation of the Travelling Salesman Problem (TSP), OptiTour is capable of determining the optimal route that covers multiple destinations at the least cost. The system also features a user-friendly Graphical User Interface (GUI), developed using Flask, HTML, CSS, and JavaScript, to visualize the optimal routes in a way that is easy to understand.

The rest of the paper will delve into the related works, the details of our solution, and the data structures and algorithms used. We will also provide a comprehensive comparison of our solution with existing technologies, a detailed analysis of our results, and a discussion of potential future improvements.

## 2 Related Works

### 2.1 Literature Review

A relevant study in [1] explores the concept of a flexible bus route optimization scheduling model that considers dynamic changes in passenger demand. The proposed model showed promising results in a simulation, reducing the planning path error, shortening travel distance and passenger travel time. The study in [2] presents two strategies for computing travel plans in public transportation systems, considering route constraints. The first uses connectivity matrices to identify feasible plans, while the second relies on hubs where service routes converge. Both strategies performed well in field tests.

Zar and Sein [3] tackles the traffic problem in Yangon area which has negatively impacted citizens' quality of life due to pollution, time wastage, and reliance on hydro-carbon fuel. It focuses on algorithms to provide bus information to promote public transportation and reduce traffic demands. Dijkstra and A\* algorithms are highlighted as efficient ways to determine the shortest routes. Chien et al. [4] proposes a model for optimizing bus routes in public transit systems, focusing on areas with a commuter travel pattern. The model minimizes total costs while considering diagonal links in the network. It transforms the network into a pure grid for application to irregular grid networks, beneficial for planning and evaluating bus services in many cities.

Popular navigation apps exist such as Google Maps and Waze. However, these tools primarily focus on individual vehicle navigation and do not cater to the specific needs of a travel bus business operation. Furthermore, the solutions are not tailored to the traffic environment in Singapore and thus do not calculate them based on ERP costs, instead focusing on shortest travel time based on traffic conditions. Dedicated logistics management solutions also exist but often lack user-friendly interfaces and might not include all the factors affecting a tourist bus service [5, 6].

### 2.2 Tools and Technologies

The OptiTour will be built utilizing a number of technologies and methodologies to ensure robustness, reliability, and scalability. Specially, the OptiTour will employ Python as the primary language for implementation [7], OSMnx to retrieve and manipulate the street network data required for the dynamic scheduling system [8], NetworkX to provides a wide range of tools and algorithms for working with networks and graph structures, LTA DataMall to provide APIs for real-time vehicle locations and other transport-related data in Singapore and essential data inputs such as Electronic Road Pricing (ERP) data [9], Flask for the UI implementation [10], Matplotlib in the Flask application to visualize the route and save it as an image file, and Pandas for efficient data manipulation and analysis [12].

## 3 Solution Designed

Our solution will be tailored to the needs of travel operators in Singapore and will thus calculate the best routes based on costs, factoring the costs of both ERP Gantries and fuel mileage with distance into the route calculations. This is implemented using

the Dijkstra’s algorithm which finds the shortest path between nodes in a graph. This algorithm uses weights of the edges to find the path that shortens the total distance between the starting node and the next following node.

### 3.1 System Diagram

The flow of the execution of our main program described above is illustrated by Fig. 1.

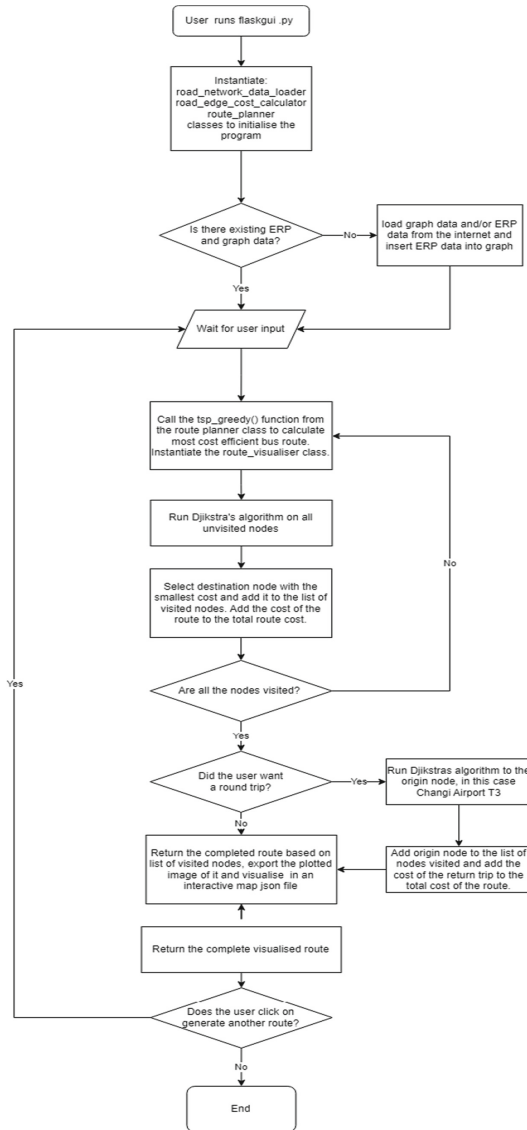


Fig. 1. A flowchart diagram of the main program

The solution will be able to plot routes starting at Changi Airport Terminal 3 to a list of hotel locations in Singapore. These hotel destinations can be arranged by the user after selecting from a set pool of hotel destinations. The hotel destinations are chosen based on their ratings, frequency of being fully booked and relative distance to each other. As Changi Airport Terminal 3 services many intercontinental flights, the hotels selected will cater to international tourists looking for luxury 5-star hotel accommodations in Singapore. The application will allow users to select from a list of the 20 5-star hotels in Singapore chosen based on how remote they are from other hotels and allow them to get a route from Changi Airport Terminal 3 to any number of them in any order.

List of hotels: Marina Bay Sands, Ritz Carlton, Shangri-La (main), Park Royal, Mandarin Oriental, Capella, Dusit Thani Laguna Singapore, Hotel Indigo Singapore Katong, The Clan, Vagabond, Duxton Reserve Singapore, W Singapore, The St. Regis Singapore, Singapore Marriott Tang Plaza Hotel, M Social Singapore, The Fullerton Hotel, Four Seasons Hotel, InterContinental Singapore, Raffles Singapore, Sofitel Singapore, Andaz Singapore.

### 3.2 Data Structures Used

**Dictionary.** Dictionaries were used in our pathing algorithms as far as possible due to their low time complexity of access. The nature of our data set also suited this as there are fixed data for nodes and edges.

**Minimum heap.** The minimum heap is a tree data structure suited to accessing the minimum of a dataset with a low time complexity. We utilized the python `heapq` library to help us implement this. It is thus suited to our implementation of Dijkstra's algorithm, ensuring it performs at  $O(V + E \log V)$  complexity in the worst case. Figure 2 (a) illustrates the layout of a minimum heap which allows for easy access to the minimum value in a heap of values.

**Multi-directional graph.** Our road network is visualized using the Multi-directional graph data structure from the library `NetworkX`, with street data from `OSMnx`. The edge data contains attributes of the road segment, such as length, ERP rates, etc. which aid in the calculation of the best route. In our graph, road intersections are represented as nodes while roads are represented as edges. Figure 2 (b) illustrates the various functionalities that a multi-direction graph is capable of that made it suitable for use in OptiTour.

## 4 Solution Implementation

### 4.1 Classes and Methods

The system uses class objects and their methods to organize various processes in the algorithm's execution.

**RoadNetworkDataLoader:** It contains methods that detect and load the graph and ERP data and download them if they are missing. The class provides methods to check the presence of the graph file, load or retrieve the road network graph, save the graph, load

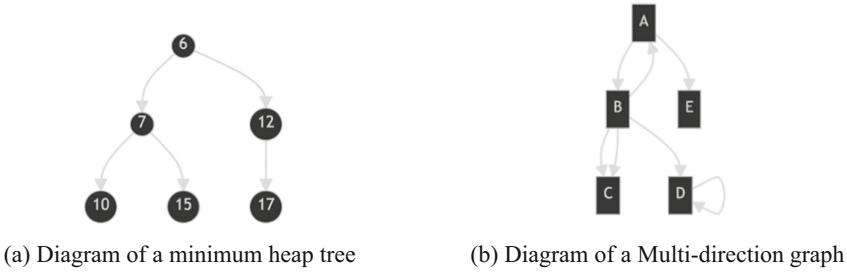


Fig. 2. Data structures

ERP data, and load coordinate data as a DataFrame for easy access by other methods later.

**RoadEdgeCostCalculator:** The RoadEdgeCostCalculator class is responsible for calculating the cost of traversing a road edge in a road network. It considers two factors: fuel consumption and ERP charges. The class provides methods to insert ERP rates into the road network graph and calculate the cost of traversing a road edge based on the provided attributes.

**RoutePlanner:** The RoutePlanner class is designed to plan routes in a road network using a greedy approach to solve the TSP. The class takes into account a graph representing the road network, an edge cost calculator for calculating road edge costs, and an origin node to start the path. The class provides a method to solve the TSP using a greedy approach.

**RouteVisualizer:** The RouteVisualizer class is responsible for visualizing and manipulating routes in a road network. It provides methods to plot the route on a map, save the route as a PNG file, retrieve the road names along the route, and convert the route to GeoJSON format used to display an interactive map in the GUI’s output.

**DijkstraHeap:** The DijkstraHeap class implements Dijkstra’s algorithm for finding the shortest path in a graph. As the name implies, the minimum heap data structure is used to decrease the time complexity of accessing the edge with the minimum cost in the implementation of Dijkstra’s algorithm.

### 4.2 ERP Implementation

ERP implementation is a crucial component of the algorithm, as it allows for the consideration of ERP charges in the cost calculation process. The algorithm inserts ERP rates and zone information into the graph based on ERP gantry coordinates. For each road segment, the algorithm finds the corresponding ERP rate within the specified time range and associates it with the segment. This enables the calculation of accurate costs, considering both fuel consumption and ERP charges.

**ERP Data Retrieval:** The ERP data is obtained from a reliable data source, specifically the LTA DataMall API. The algorithm accesses the ERP rates and zone information through the API using the appropriate authentication headers. The retrieved data is in JSON format.

**Data Integration and Preprocessing:** The ERP data is integrated into the road network graph. Each ERP gantry's location is matched with the nearest road edge in the graph using the nearest edges function from the OSMnx library. The corresponding zone identifier is assigned to the road edge.

**ERP Charges Assignment:** For each road edge associated with an ERP gantry, the ERP rates and time ranges are assigned as attributes to the graph edge. The ERP charges are stored in a dictionary structure within the edge attributes, categorized by day type and time range.

**Cost Calculation:** During the route planning process, the algorithm calculates the cost of traversing each road edge. It takes into account both the fuel consumption and the ERP charges. The fuel consumption cost is calculated based on the length of the edge and the current fuel price per liter. The ERP charges are retrieved from the edge attributes and added to the total edge cost if applicable.

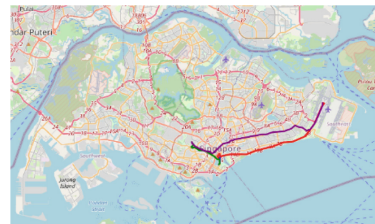
**Cost Optimization:** The integration of ERP charges enables the algorithm to optimize routes based on overall cost, including both fuel consumption and ERP charges. By considering ERP charges in the cost estimation process, the algorithm can identify routes with lower overall costs and guide users towards more cost-effective paths.

### 4.3 GUI Implementation

A GUI is essential to keep our application usable for as many people as possible. It is implemented as a flask web application in HTML and CSS with more advanced features being realized in JavaScript. It can show the route in brief on a zoomed-out map as shown in Fig. 3 (a). It can also show an interactive map view like the one in Fig. 3 (b) in a lightbox by clicking on the map.



(a) Diagram of the zoomed-out map



(b) Diagram of the interactive map

**Fig. 3.** GUI diagrams

The found route is also displayed as a breadcrumb trail of streets as shown in Fig. 4. Relevant data about the route such as estimated cost, ERPs passed, and distance are also displayed as in Fig. 5.

JavaScript is used on the inputs which allows for multiple fields to be added and removed depending on how many destinations the user wants. Checking is also performed to ensure that no duplicate hotels can be selected and input. The functionalities enabled by JavaScript for adding and removing fields are shown in Fig. 6.

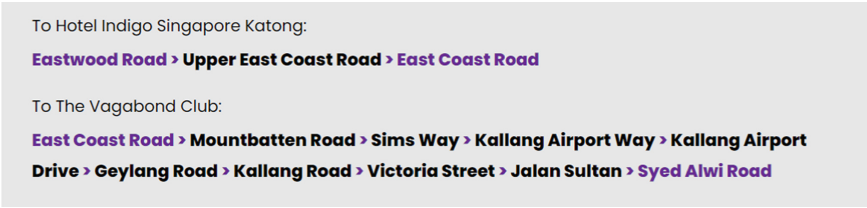


Fig. 4. Diagram of the route displayed

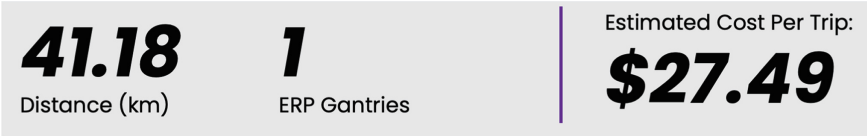


Fig. 5. Diagram of the route displayed

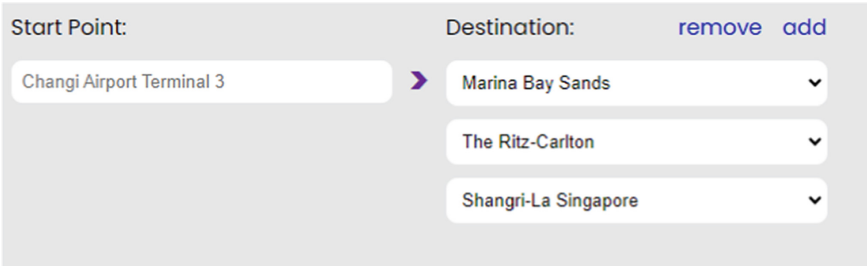


Fig. 6. Diagram of the user input

### 4.4 Dataset Used

The algorithm relies on three primary data sources: OpenStreetMap data, ERP rates data from LTA DataMall, and coordinates of each ERP Gantry from Google Maps. OpenStreetMap data provides detailed information about the road network, including road types, connectivity, and geographical coordinates. This data is used to construct the road network graph, which serves as the foundation for calculating least-cost paths. ERP rate data contains information about the rates charged for traversing specific zones at different timings. This data is used to determine the ERP charges associated with each road segment in the graph at all times of day. We also generated a spreadsheet of coordinates of the destinations and ERPs using longitude and latitude data found on Google Maps.

### 4.5 Data Pre-processing

To determine the nodes that will be associated with our destination coordinates, we converted the coordinates to points and used the NetworkX and OSMnx functions `osmnx.get_nearest_node()` and `osmnx.get_nearest_edge()`. These ERP rate values are

then inserted into the OpenStreetMap graph at the edges found earlier to be accessed by the algorithm calculating the most cost-effective route. The destination node numbers and names found earlier are stored in a dictionary named “all\_destination\_ids\_dict” and “all\_destination\_names\_dict” for easy translation between destinations and node numbers. The average fuel consumption of the buses is around 25 L per 100 km [13].

## 5 Results and Insight

We implemented the following algorithms: Dijkstra’s algorithm, A-Star algorithm and the Bellman-ford algorithm. We benchmarked the 3 algorithms against each other and noted their performance over 20 runs with values and 1 calibration run per task. Each run was performed with fixed destination and origin nodes, namely, the first and last nodes in our graph. The benchmark results shown in Table 1 shows the total duration for each task, which includes the time taken to run the task multiple times, as well as warmup runs and loop iterations.

In general, the Bellman Ford algorithm took longer to execute compared to the other two. Between Dijkstra and A-Star algorithms, we have chosen to incorporate Dijkstra over A-Star due to several reasons. A-Star relies on heuristics and making heuristic selection can be challenging and computationally expensive. Dijkstra, on the other hand, does not require heuristics, simplifying implementation. A-Star’s additional memory requirements and inefficiency in uniform cost graphs further contributed to our decision. Considering the project’s goals and constraints, Dijkstra’s algorithm’s simplicity, lower memory usage, and ability to guarantee the shortest path without heuristics make it the optimal choice for our pathfinding needs.

**Table 1.** Shortest path algorithms comparisons.

Benchmark time complexity	A-Star	Dijkstra’s	Bellman-ford
Total duration	148.8 s	149.6 s	183.1 s
Mean performance	1.78 s	1.79 s	2.20 s
Median performance	1.78 s	1.77 s	2.20 s
Standard deviation	0.06 s	0.06 s	0.02 s
Minimum	1.69 s	1.71 s	2.16 s
Maximum	1.91 s	1.92 s	2.27 s

The TSP was also implemented as it allows for our algorithm to run on multiple destinations regardless of what order they are entered in and calculate the shortest round-trip route which would likely be used for a shuttle bus service. The built in NetworkX TSP function took too long to run due to our large node dataset so we decided to code a custom implementation of the TSP that only considered our destination nodes.



The brute force method, Dynamic Programming, the Greedy method and Without TSP algorithm are evaluated and the results are shown in Table 2. Compared to Without TSP algorithm, the TSP has greater time complexity on average due to having to find the node with the lowest cost from the current position. However, the routes generated are also less efficient on average if an optimal sequence of destinations is not entered. This inefficiency is caused by the suboptimal route having cases where it back tracks to a destination previously bypassed by it. Thus, the trade-off between time complexity of the algorithms and the accuracy of the shortest route between multiple destinations is justified. This algorithm will thus be able to use Dijkstra's Algorithm in the TSP to give the user the most cost-effective path in terms of fuel and ERP costs to their set of destinations in a reasonable amount of time.

**Table 2.** Shortest TSP comparison.

Algorithm	Time complexity	Pros	Cons
Brute force	$O(n!)$	Always finds the optimal solution	Not feasible for large $n$ due to high time and space complexity
Dynamic programming	$O(n^2 2^n)$	More efficient than brute force for larger $n$ , always finds the optimal solution	High time and space complexity, not feasible for very large $n$
Greedy	$O(n^2)$	Efficient for larger datasets	Does not always guarantee the globally optimal solution
Without TSP	$O(n)$	Much smaller time complexity overall	Does not generate the most cost-effective path if the sequence of destinations is not optimal

Our algorithm is tailored for the needs of a bus operator as it can integrate ERP charges at all times of day into its cost calculation process and offers the capability to calculate the best route between multiple destinations, with the option of making it a round trip. It is also capable of displaying a detailed route with an intuitive breadcrumb trail of road names to the destination. These quality-of-life improvements are currently unavailable in similar map software such as Google Maps and Waze.

We have concluded that the optimum algorithm for calculating a cost-effective route when applied to the TSP is Dijkstra's algorithm but only when implemented with the minimum heap data structure. This will result in a good balance between time complexity and accuracy of results generated. This is also supported by how most other similar solutions in the market implement Dijkstra's algorithm in their applications. The TSP

implementation, however, is more specific to our chosen application, as bus routes may be required to be round trips and have to optimise routes between multiple nodes in the optimum order. Our analysis of the growth of the time complexity of our TSP algorithm is shown in Table 3.

**Table 3.** Time complexity of TSP algorithm based on number of destinations.

Input size (number of destinations)	Time complexity (s)
1	0.1
5	0.7
10	2.4
20	8.6

## 6 Conclusion

OptiTour's combination of an easy-to-use GUI and relevant considerations of the needs of bus operators allow it to be uniquely suited to the road network in Singapore. The use of Dijkstra's algorithm with the TSP allows for routes to any number of destinations to be efficiently plotted with minimal increase in runtime while comprehensively considering all routes. Those routes are reordered by our solution to generate the order resulting in the best possible route between all selected destinations.

This software could be expanded to assist in countrywide bus route planning by allowing it to take in start points from a list of start points. It could also be made to estimate route times by taking traffic conditions into consideration, avoiding stretches of road that have higher chances of traffic jams.

## References

1. Sun, J., Chen, Y., Huang, J., Wei, P., Song, C.: Flexible bus route optimization scheduling model. *Adv. Civ. Eng.* **2021** (2021)
2. Liu, C.-L., Pai, T.-W., Chang, C.-T., Hsieh, C.-M.: Path-planning algorithms for public transportation systems. In: *Proceedings on 2001 IEEE Intelligent Transportation Systems*, Oakland, CA, USA (2001)
3. Zar, M.T., Sein, M.M.: Using A\* algorithm for public transportation system in Yangon Region. *Int. J. Adv. Comput. Eng. Netw.* **3**(3), 7–10 (2015)
4. Chien, S.I.-J., Dimitrijevic, B.V., Spasovic, L.N.: Optimization of bus route planning in urban commuter networks. *J. Public Transp.* **6**(1), 53–79 (2003)
5. Google Maps. <https://www.google.com/maps/about/#/>. Accessed 25 May 2023
6. Waze: Driving Directions, Traffic Reports & Carpool Rideshares. <https://www.waze.com/about/>. Accessed 25 May 2023
7. Python.org: Welcome to Python.org, 22 May 2023. <https://www.python.org/>. Accessed 25 May 2023

8. Boeing, G.: OSMnx. <https://github.com/gboeing/osmnx>. Accessed 25 May 2023
9. Land Transport Authority (LTA): DataMall. <https://DataMall.lta.gov.sg/content/DataMall/en.html>. Accessed 25 May 2023
10. Flask: Welcome to Flask—Flask Documentation (2.3.x). <https://flask.palletsprojects.com/en/2.3.x/>. Accessed 25 May 2023
11. Babitz, K.: Introduction to Plotting with Matplotlib in Python (2023). <https://www.datacamp.com/tutorial/matplotlib-tutorial-python>. Accessed 25 May 2023
12. Pandas - Python Data Analysis Library (n.d.). <https://pandas.pydata.org/>. Accessed 25 May 2023
13. Eco driving quality system for city and intercity buses. <https://jv-technoton.com/cases/eco-driving-quality-system-for-city-and-intercity-buses/>. Accessed 25 May 2023