TOUR ROUTE OPTIMISER USING TSP HEURISTIC ALGORITHMIC APPROACH

CSD415 Project Phase 1

CS20D04 CHN20CS006 Abin Skaria

CS20D32 CHN20CS056 Jais Tomy

CS20D47 CHN20CS093 Sahal Basheer

CS20D50 CHN20CS100 Shiwine K Sebastian

CS20D53 CHN20CS104 Souhrid Suresh

B. Tech. Computer Science & Engineering

Department of Computer Engineering

College of Engineering Chengannur Kerala - 689121

Phone: (0479) 24541125 Fax: (0479) 2451424

Website: www.ceconline.edu

2023

DEPARTMENT OF COMPUTER ENGINEERING

COLLEGE OF ENGINEERING CHENGANNUR

2023

CERTIFICATE

This is to certify that the report **TOUR ROUTE OPTIMISER USING TSP HEURISTIC ALGORITHMIC APPROACH** submitted by **Abin Skaria, Jais Tomy, Sahal Basheer, Shiwine K Sebastian, Souhrid Suresh**, Seventh Semester B.Tech. Computer Science & Engineering students, for the course work in **CSD415 Project Phase I**,under our guidance and supervision, in partial fulfillment of the requirements for the award of the degree, B. Tech. Computer Science & Engineering of **APJ Abdul Kalam Technological University.**

Smt. Syeatha Merlin Thampy
(Project Guide)
Assistant Professor,
Dept. of Computer Engineering
College of Engineering,
Chengannur

Prof. Ahammed Siraj K K
(Project Coordinator)
Associate Professor,
Dept. of Computer Engineering
College of Engineering,
Chengannur

Dr. Manju S Nair
Head of the Department,
Assistant Professor,
Computer Engineering
College of Engineering,
Chengannur

December 15, 2023

Acknowledgement

We take this opportunity to express our deepest sense of gratitude and sincere thanks to everyone who helped us complete this work successfully. We express our sincere thanks to **Dr. Manju S Nair**, Head of Department, Department of Computer Engineering, College of Engineering Chengannur for providing us with all the necessary facilities and support.

We would like to express our sincere gratitude to **Prof. Ahammed Siraj K K**, Associate Professor, Department of Computer Engineering, College of Engineering, Chengannur for their support and cooperation.

We would like to place on record our sincere gratitude to our Project guide, **Smt. Syeatha Merlin Thampy**, Assistant Professor, Department of Computer Engineering, College of Engineering, Chengannur for the guidance and mentorship throughout the course.

Abin Skaria, Jais Tomy, Sahal Basheer, Shiwine K Sebastian, Souhrid Suresh

Abstract

Tourism is a thriving industry that contributes significantly to the global economy. As tourists seek best experiences, the need for efficient itinerary planning becomes paramount. This project explores the application of the Traveling Salesman Problem (TSP) in the context of tourism to optimize tourist itineraries. The primary objective of this project is to develop a software that assists tourists in creating optimal travel routes by solving the TSP. In the context of tourism, this problem translates to finding the most efficient route that allows tourists to explore a destination's key attractions while minimizing travel time. Our approach involves collecting and curating data on tourist attractions and distances. The TSP algorithm is then applied to this dataset to generate optimized itineraries. The software will provide users with the flexibility to customize their preferences, such as prioritizing certain attractions and accommodating time constraints.

Contents

List of Figures

List of Tables

1. Introduction

Tourism Planner is an innovative and user-friendly platform designed to simplify and enhance the travel planning experience. Seamlessly integrating cutting-edge technology, it offers a dual-sided interface catering to both administrators and users. Administrators play a pivotal role by providing and updating essential travel data, while users leverage the system to access, customize, and visualize information through an interactive map. Facilitated by a central server, Tourism Planner fosters collaboration, ensuring real-time data synchronization and scalability. This dynamic platform empowers users to plan their journeys efficiently and contribute to the collective enrichment of the travel community.

1.1 Proposed Project

In the dynamic world of global tourism, our project redefines travel planning by addressing the challenges through the renowned Traveling Salesman Problem (TSP). This combinatorial optimization tool finds the shortest route and ensuring efficiency. Our sophisticated software seamlessly integrates comprehensive data on attractions, distances, utilizing the TSP algorithm to craft meticulously optimized itineraries. Designed for flexibility, users can customize their adventures, prioritizing attractions and managing time constraints.

1.1.1 Problem Statement

The project aims to develop a software using the Traveling Salesman Problem (TSP) to assist tourists in creating optimal travel routes. This software aims to improve the efficiency of exploring tourist attractions.

1.1.2 Proposed Solution

- **TSP Application:** Apply the TSP algorithm to optimize tourist routes. TSP aims to find the shortest route visiting locations exactly once and returning to the starting point.
- **Project Focus:** Develop a software as the primary project focus. Assist tourists in creating optimal travel routes by solving the TSP.
- **Combinatorial Optimization:** TSP is a well-known combinatorial optimization problem. It is adapted for tourism to minimize travel time.
- **Data Collection and Curation:** Collect and curate data on tourist attractions and distances.Input this dataset into the TSP algorithm for itinerary optimization.
- **User Flexibility:** Provide users with flexibility in customizing preferences.Options include prioritizing attractions and accommodating time constraints.
- **Software Benefits:** Enhance user experience by offering streamlined travel planning.Cater to the diverse needs of tourists seeking personalized and efficient travel routes.

2. Report of Preparatory Work

2.1 Literature Review

2.1.1 IntechOpen, Travelling Salesman Problem [1]

The idea behind TSP to consider a problem from the everyday life from a mathematical point of view. A traveling salesman has to visit exactly once each one of a list of m cities and then return to the home city. He knows the cost of traveling from any city i to any other city j. In this the problem of finding algorithmic technique leading to good/optimal solutions for TSP (or for some other strictly related problems) is considered. TSP is a very attractive problem for the research community because it arises as a natural subproblem in many applications concerning the every day life. Indeed, each application, in which an optimal ordering of a number of items has to be chosen in a way that the total cost of a solution is determined by adding up the costs arising from two successively items, can be modelled as a TSP instance. Thus, studying TSP can never be considered as an abstract research with no real importance.

2.1.2 On solving TSP by genetic algorithms [2]

Presenting a genetic algorithm for solving the traveling salesman problem using genetic algorithms to optimality for traveling salesman problems with up to cities. Proposed a genetic algorithm for the traveling salesman problem that generates very good but not optimal solutions for traveling salesman problems with and cities. Improved this approach by enhancing all basic components of that genetic algorithm. For our experimental investigations, used the traveling salesman problems TSP with cities, which were solved to optimality. Could solve medium-sized traveling salesman problems with up to cities in minutes average runtime on a SUN workstation. Furthermore, could solve traveling salesman problems with up to cities optimally in an acceptable time limit (e.g., the average runtime on a SUN workstation for the TSP is about minutes). The greatest examined problem with cities could be approximately solved by constructing a tour with a length percentage over the optimum.

2.1.3 An expert system for tourists using Google Maps API [3]

In the field of eTourism, it is important to present prominent objects of tourists destinations. Nowadays demand for eTourism applications is rising and the customers need rapid software development. Google Maps API is a technology provided by Google based on AJAX, which powers many map-based services. This paper presents an expert system for tourists. The realized software uses free, public API service from Google Maps. The system utilizes a knowledge base formed by tracking user actions. The expert module suggests information of special interest to the user.

2.2 Algorithm Analysis

The project mainly focuses on Travelling Salesman Problem(TSP).The Traveling Salesman Problem is a well-known optimization challenge where the objective is to determine the shortest possible route that visits each city exactly once and returns to the starting city. Represented as a complete graph with cities as nodes and distances as edges, TSP seeks to minimize the total traveled distance. With applications in logistics, transportation, and manufacturing, TSP is NP-hard, meaning solving it efficiently for all cases is computationally challenging. Various approaches, from brute-force enumeration to heuristic algorithms like nearest neighbor and genetic algorithms, are employed to find optimal or near-optimal solutions, making TSP a central problem in the study of combinatorial optimization.There are many algorithms based on Travelling Salesman Problem.So from that we have to analyze all the TSP algorithm and select the best approximation and optimization algorithm to implement on project that has less time complexity.

Using both optimization and approximation algorithms for finding the shortest path allows for a balanced approach that considers both efficiency and speed in various scenarios. Optimization algorithms aim to find the most accurate solution, but they may be computationally expensive. On the other hand, approximation algorithms provide quicker solutions that may not be optimal but are close enough. By employing both types of algorithms, it can dynamically choose the appropriate method based on the specific requirements of the task and the available computing resources. This hybrid approach provides flexibility to balance the trade-off between accuracy and computational efficiency, ensuring a more responsive and adaptable system for finding the shortest path.

So, there are many algorithms coming under both approximation and optimization algorithms so we are going to analyze some of them. Below are the following algorithms which are to be analyzed.

1. **Cheapest Insertion :** The cheapest insertion algorithm starts with a partial tour and incrementally adds the remaining cities to the tour. At each step, the algorithm selects the city that results in the least increase in the total tour cost when inserted into the current tour. The cost is typically defined as the distance between the inserted city and its two neighboring cities in the tour. While the solution generated by the cheapest insertion algorithm may not be optimal, it provides a reasonably good approximation in a computationally efficient manner.

2. **Christofides Algorithm :** The Christofides algorithm begins by constructing a minimum spanning tree connecting all cities. It then finds a minimum weight perfect matching for the odd-degree vertices in the tree. Combining the spanning tree and the matching results in an Eulerian circuit, which is shortened to create a Hamiltonian circuit. The final circuit provides a solution to the Traveling Salesman Problem, guaranteeing a tour length within 3/2 times the optimal solution for metric instances.

3. **Clark Wright :** The Clark-Wright Savings Algorithm is a heuristic method for solving the Vehicle Routing Problem. It starts by computing the savings for each pair of customers, representing the potential cost reduction by combining their routes. The algorithm then sorts the savings in descending order and iteratively merges routes, considering the highest savings first. This process continues until all customers are visited, resulting in a set of optimized routes for the vehicles.

4. **Farthest Insertion :** The Farthest Insertion Algorithm is a heuristic for solving the Traveling Salesman Problem (TSP). It begins by selecting an initial city arbitrarily and then identifies the city farthest from the current tour. The chosen city is inserted into the tour at the position where it causes the least increase in total distance. This process is repeated until all cities are included, creating

a near-optimal solution that is often more efficient than random or sequential insertion methods.

5. **Greedy :** A greedy algorithm is an approach to problem-solving that makes locally optimal choices at each stage in the hope of finding a global optimum. It selects the best available option at each step without reconsidering previous choices. Greedy algorithms are often used for optimization problems and are characterized by their simplicity and efficiency. While they don't guarantee the optimal solution in every case, greedy algorithms are valuable for their speed and ease of implementation.

6. **Nearest Insertion :** The Nearest Insertion Algorithm is a heuristic method for solving the Traveling Salesman Problem (TSP). It starts by selecting an initial city and then repeatedly adds the city that is closest to the current tour. The chosen city is inserted into the tour at the position where it results in the least increase in total distance. This process continues until all cities are included, providing a solution that is often close to optimal in a computationally efficient manner.

7. **Nearest Neighbour :** The Nearest Neighbour Algorithm is a heuristic for solving the Traveling Salesman Problem. It starts by selecting an initial city and repeatedly chooses the closest unvisited city to the current location. The chosen city is added to the tour, and the process continues until all cities have been visited, forming a circuit. While the Nearest Neighbour Algorithm is easy to implement and computationally efficient, it may not always produce an optimal solution due to its greedy nature.

8. **Random Insertion :** The Random Insertion Algorithm is a heuristic approach for solving the Traveling Salesman Problem (TSP). It begins by randomly selecting a city as the starting point. At each step, it chooses a remaining city at random and inserts it into the current tour at a position where it causes the least increase in total distance. This process is repeated until all cities are included, generating a tour that, while not guaranteed to be optimal, provides a solution that can vary based on the random selection of starting points and insertions.

9. **2opt :** The 2-opt algorithm is a local search heuristic used for optimizing solutions in optimization problems such as the Traveling Salesman Problem

(TSP). It starts with an initial solution and iteratively improves it by swapping pairs of edges in the tour to reduce the overall tour length. The algorithm continues to perform these edge swaps until no further improvement can be made. While not guaranteed to find the global optimum, the 2-opt algorithm is efficient and often produces good solutions for TSP instances.

10. **3opt :** The 3-opt algorithm is an extension of the 2-opt algorithm, commonly used for optimizing solutions in problems like the Traveling Salesman Problem (TSP). It involves iteratively exchanging three edges in a tour to find a shorter overall tour length. The algorithm explores various combinations of edge swaps to improve the solution until no further enhancements can be made. Although computationally more intensive than 2-opt, 3-opt has the potential to find better solutions by considering a broader range of possibilities for edge exchanges in the tour.

11. **Ant Colony :** The Ant Colony Optimization (ACO) algorithm is a nature-inspired optimization technique often used for solving combinatorial optimization problems. It is based on the foraging behavior of real ants. Artificial ants construct solutions to problems by depositing pheromone trails on paths, with the intensity of the pheromone influencing subsequent ant choices. Over time, paths with stronger pheromone concentrations are more likely to be chosen, allowing the algorithm to converge towards optimal or near-optimal solutions. The ACO algorithm is particularly effective in solving problems such as the Traveling Salesman Problem and the Job Scheduling Problem.

12. **Brute Force :** The Brute Force algorithm is a straightforward method for solving problems by systematically examining all possible solutions. It exhaustively checks each solution candidate to find the optimal one. While it guarantees finding the optimal solution, it can be computationally expensive, especially for large problem instances, as it evaluates all combinations. Brute Force is often used as a benchmark to compare the efficiency of more sophisticated algorithms for solving optimization problems.

13. **Chained Lin Kernighan :** The Chained Lin-Kernighan (CLK) algorithm is an iterative optimization algorithm commonly applied to solve the Traveling Salesman Problem (TSP). It is an enhancement of the Lin-Kernighan algorithm, which is known for its effectiveness in finding high-quality solutions. CLK

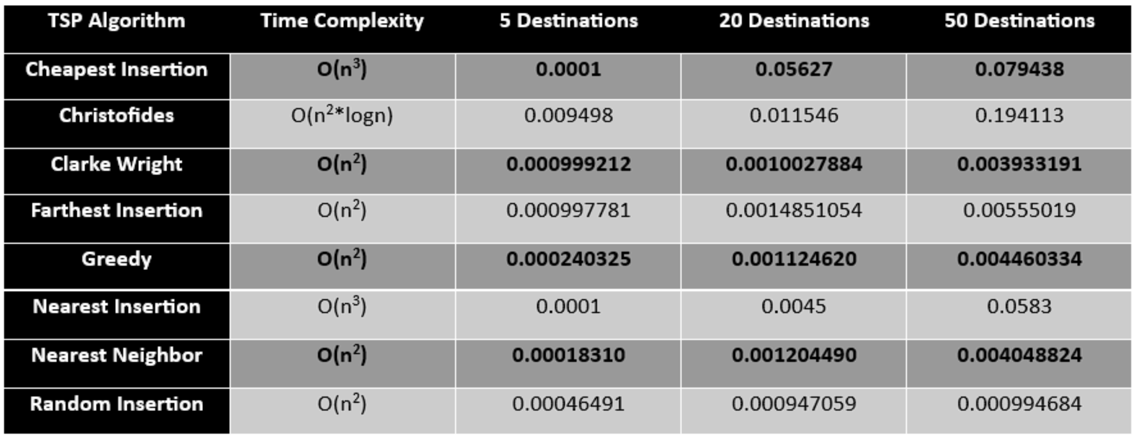
repeatedly applies the Lin-Kernighan procedure to progressively improve the current solution by iteratively making local modifications. This chaining of Lin-Kernighan steps leads to refined and optimized TSP solutions with significantly reduced tour lengths.

14. **Dynamic programming :** Dynamic Programming is a technique used to solve optimization problems by breaking them down into overlapping subproblems and solving each subproblem only once, storing the results for future reference.

15. **Lin Kernighan :** The Lin-Kernighan Algorithm is a heuristic optimization method designed to solve the Traveling Salesman Problem (TSP). It works by iteratively improving an initial solution through a sequence of local search operations. The key operation involves breaking and reconnecting tour segments to create new candidate solutions and accepting those that lead to a decrease in the total tour length. Lin-Kernighan is known for its efficiency and ability to find high-quality solutions for the TSP, making it widely used in practice.

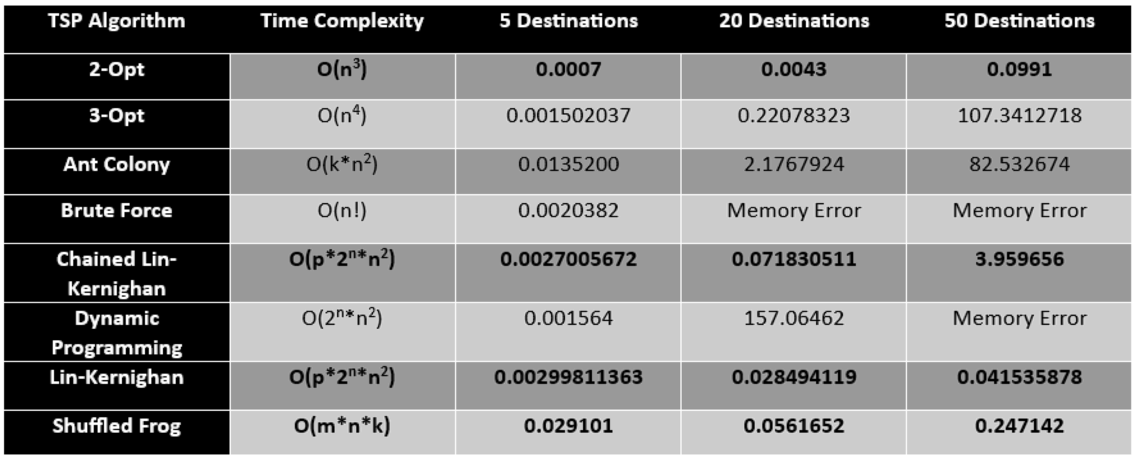
16. **Shuffled Frog :** The Shuffled Frog-Leaping Algorithm (SFLA) is a nature-inspired optimization algorithm modeled after the cooperative foraging behavior of frogs. The algorithm involves partitioning a population of “virtual frogs” into groups, allowing them to iteratively explore solution spaces, and exchanging information through a shuffling process. SFLA has been employed in various domains to find high-quality solutions to optimization problems, leveraging the collaboration and adaptation inherent in its frog-inspired design.

In Table 2.1 and Table 2.2, both show the execution time based on random input distances for 5, 20, and 50 destinations for all the algorithms. Table 2.1 presents data for the approximation algorithms, while Table 2.2 provides information for the optimization algorithms. We analyze and shortlist the algorithms based on the execution time. The highlighted algorithms in the table are the ones selected for further analysis.



TSP Algorithm	Time Complexity	5 Destinations	20 Destinations	50 Destinations
Cheapest Insertion	**O(n^3)**	**0.0001**	**0.05627**	**0.079438**
Christofides	O(n^2*logn)	0.009498	0.011546	0.194113
Clarke Wright	**O(n^2)**	**0.000999212**	**0.0010027884**	**0.003933191**
Farthest Insertion	O(n^2)	0.000997781	0.0014851054	0.00555019
Greedy	**O(n^2)**	**0.000240325**	**0.001124620**	**0.004460334**
Nearest Insertion	O(n^3)	0.0001	0.0045	0.0583
Nearest Neighbor	**O(n^2)**	**0.00018310**	**0.001204490**	**0.004048824**
Random Insertion	O(n^2)	0.00046491	0.000947059	0.000994684

Table 2.1: Execution time taken for Approximation Algorithms



TSP Algorithm	Time Complexity	5 Destinations	20 Destinations	50 Destinations
2-Opt	**O(n^3)**	**0.0007**	**0.0043**	**0.0991**
3-Opt	O(n^4)	0.001502037	0.22078323	107.3412718
Ant Colony	O(k*n^2)	0.0135200	2.1767924	82.532674
Brute Force	O(n!)	0.0020382	Memory Error	Memory Error
Chained Lin-Kernighan	**O(p*2^n*n^2)**	**0.0027005672**	**0.071830511**	**3.959656**
Dynamic Programming	O(2^n*n^2)	0.001564	157.06462	Memory Error
Lin-Kernighan	**O(p*2^n*n^2)**	**0.00299811363**	**0.028494119**	**0.041535878**
Shuffled Frog	**O(m*n*k)**	**0.029101**	**0.0561652**	**0.247142**

Table 2.2: Execution time taken for Optimization Algorithms

From the above two tables, we can observe the shortlisted algorithms for approximation: Cheapest Insertion, Clarke Weight, Greedy, Nearest Neighbor; and for optimization algorithms: 2-Opt, Ant Colony, Chained Lin-Kernighan, Lin-Kernighan. We further analyze these algorithms by using the same input distances for 5, 10, 20, and 50 destinations for each algorithm. Based on the obtained execution time and distances, we analyze and refine our shortlist of algorithms. Tables 2.3 and 2.4 present the execution time and distances for each algorithm.

TSP Algorithm	Time Complexity	Metric	5 Destinations	10 Destinations	20 Destinations	50 Destinations
Cheapest Insertion	$O(n^3)$	Distance Covered	245.90	378.1736	455.06667	530.9390
		Execution Time	0.0001	0.0001	0.010035	0.05167
Clarke Wright	$O(n^2)$	Distance Covered	266.9926	348.850	454.0794	574.6649
		Execution Time	0.0002	0.0002	0.0013001	0.009248
Greedy	$O(n^2)$	Distance Covered	253.1226	317.0612	454.0794	512.5934
		Execution Time	0.0001	0.0001	0.00151685	0.005006
Nearest Neighbor	**$O(n^2)$**	**Distance Covered**	**253.1226**	**317.0612**	**454.0794**	**512.5934**
		Execution Time	**0.0001**	**0.0001**	**0.00100064**	**0.003017**

Table 2.3: Metric Analyzation for Approximation Algorithms

TSP Algorithm	Time Complexity	Metric	5 Destinations	10 Destinations	20 Destinations	50 Destinations
2-Opt	$O(n^3)$	Distance Covered	266.7095	314.6345	461.11125	455.365
		Execution Time	0.0004036	0.0027	0.0230	0.7097
Chained Lin-Kernighan	$O(p*2^n*n^2)$	Distance Covered	251.13	311.46	405.77	434.22
		Execution Time	0.00163	0.04094	2.623	120.95436
Lin-Kernighan	**$O(p*2^n*n^2)$**	**Distance Covered**	**245.90**	**285.577**	**382.71**	**457.63**
		Execution Time	**0.00041906**	**0.024599**	**0.804304**	**12.29**
Shuffled Frog	$O(m*n*k)$	Distance Covered	245.90	376.1171	690.471	1369.2154
		Execution Time	0.0243	0.0354	0.0544	0.1219

Table 2.4: Metric Analyzation for Optimization Algorithms

So after all analyzation we get into a point that Nearest Algorithm is suitable for Approximation and Lin-Kernighan is suitable for Optimization.

2.3 System Study Report

To better comprehend the technological landscape, we delved into system studies, analyzing working of various TSP algorithms. Our focus was on identifying the suitable algorithm for routing various destinations provided by user. The criteria for selecting the algorithm was, it should provide shortest route in the least time as possible. While analyzing algorithms one of the challenges we faced was when the number of destination's increased the execution time also increased considerably. So we have to select approximation algorithm to route the destinations when the number of destinations increased. Overall the system study helped us in determining the workflow behind the tourism planner.

3. Project Design

This chapter outlines the architectural blueprint for the Tourism Planner, emphasizing efficiency and user-friendliness. It covers web application architecture, hardware and software requirements, offering insights for developers and stakeholders. The clear presentation of design principles, with diagrams and detailed descriptions, lays the groundwork for effective implementation in subsequent development stages.

3.1 Web Application Architecuture: General

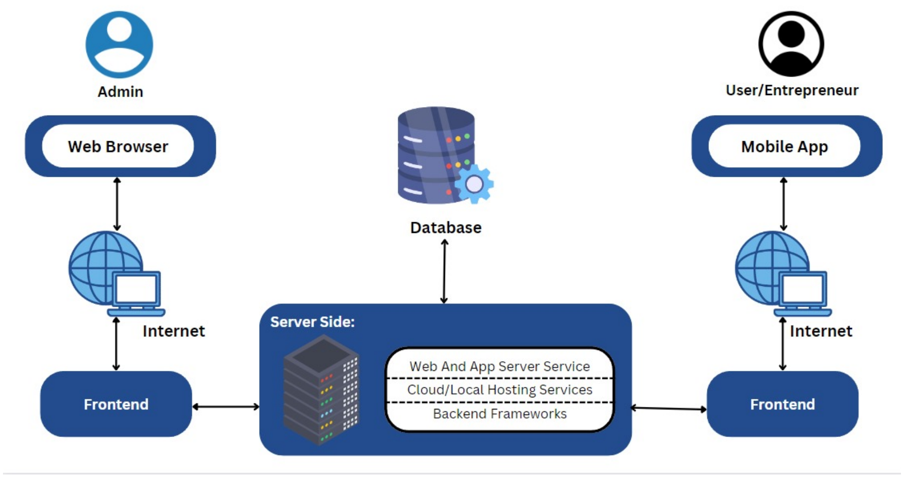


Figure 3.1: Web Application Architecture: General

The above diagram shows a web-based tourist information and planning system. Here's how it works:

1. **Client Side:**

   - User/Entrepreneur: This section caters to two user types: tourists and entrepreneurs. Tourists can use the system to plan their trips, while entrepreneurs can manage their tourism-related businesses.

- Web Browser/Mobile App: Admin can access the system through a web browser on a computer and user access a mobile app on a smartphone or tablet.

2. **Server Side:**
   - Database: This stores all the information needed for planning trips, such as tourist attractions, accommodation and reviews. It's likely a cloud-based database for scalability and accessibility.
   - Web and App Server Service: This processes user requests, retrieves data from the database, and generates responses. It could use technologies like NodeJS or Python Flask.

3. **Frontend/Backend:**
   - Frontend: This refers to the user interface that tourists and entrepreneurs interact with, likely built with frameworks like ReactJS for the web and Flutter for mobile apps.
   - Backend: This handles the behind-the-scenes logic of the system, such as data processing and communication with the database.

4. **Additional components:**
   - Internet: This connects the user's device to the server.
   - Cloud/Local Hosting Services: This refers to where the server and database are hosted. It could be a cloud platform like Amazon Web Services or Google Cloud Platform for scalability and reliability, or a local server for more control and customization.

5. **Overall flow:**
   - User Interaction: Tourists browse attractions, accommodation options and facilities. Entrepreneurs manage their listings and update information.
   - Data Retrieval: The system retrieves relevant data from the database based on user searches or filters.
   - Information Presentation: The system presents the retrieved data to the user in a user-friendly format, such as lists, maps, or detailed pages.

- Booking/Management: Tourists can book tours, accommodation, or tickets through the system. Entrepreneurs can manage their bookings and update their listings.

6. **Possible benefits:**

   - Convenience: Users can plan their entire trip in one place, from finding attractions to booking accommodation.
   - Information access: The system provides a central hub for all tourism-related information, making it easy for users to find what they need.
   - Business promotion: Entrepreneurs can reach a wider audience of potential customers through the system.

3.2 Web Application Architecture: Precise to project

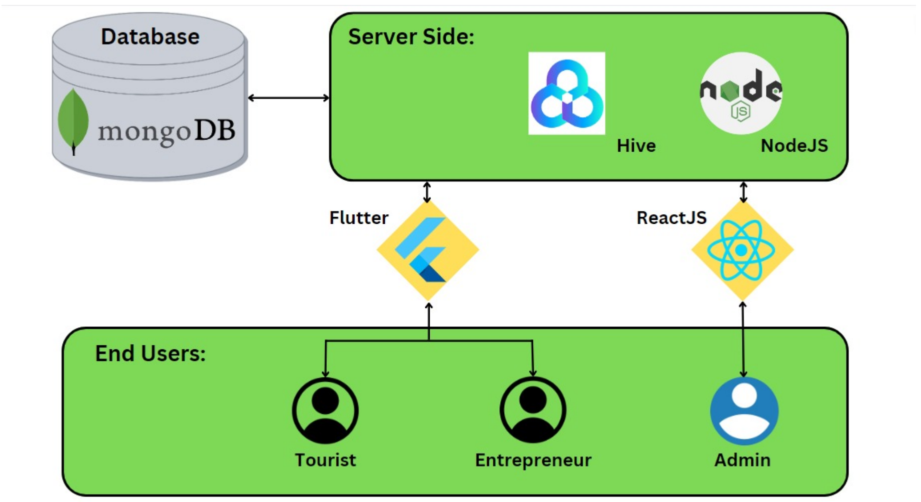


Figure 3.2: Web Application Architecture: Precise

The system is divided into two main sections: the server-side and the end-user applications.

1. **Server-side:**

   - Database: This stores all the relevant data for planning tourist routes, such as information about tourist attractions, opening hours, cost and user reviews. The preferred database is MongoDB.

- Server-side logic: This is where the system processes user queries and generates potential routes. It likely uses algorithms that consider factors like the user's preferences, travel dates, and desired activities. The server-side logic could be implemented using technologies like NodeJS.

2. **End-user applications:**

   - Flutter: This is a mobile app development framework, suggesting that the system may have a mobile app for tourists to use. The app would allow users to input their preferences and view suggested routes.
   - ReactJS: This is a JavaScript library for building user interfaces, suggesting that the system may also have a web interface. This could be useful for planning trips from a desktop computer.

3. **Overall flow:**

   - User input: The user enters their preferences and desired travel dates into the mobile app.
   - Route generation: The server-side logic processes the user input and queries the database to find potential routes that match the user's criteria.
   - Route presentation: The system presents the suggested routes to the user on the mobile app. The routes may include information about attractions, and estimated travel times.
   - User selection: The user selects a route or refines their preferences and generates new suggestions.

4. **Additional components:**

   - Hive: This is an open-source data warehouse software, suggesting that the system may use data warehousing techniques to store and organize large amounts of tourist data.
   - Node: This could refer to a general processing unit or a specific technology like NodeJS.

3.3 Algorithm Implementation

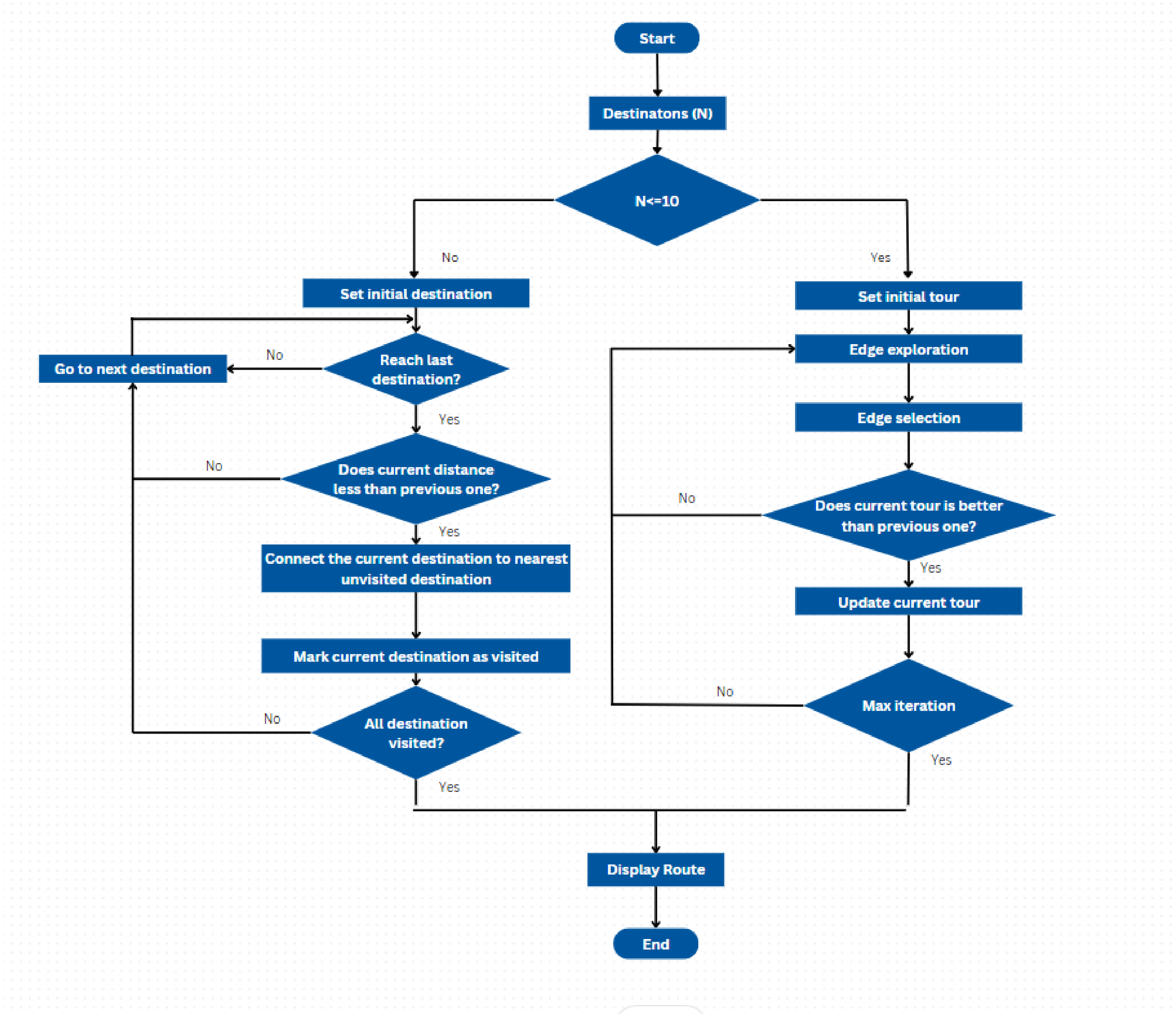


Figure 3.3: Algorithm Work Flow

3.3.1 Approximation - Nearest Neighbor

Algorithm:

Main Algorithm (nearestneighbor):

Start with an initial city (can be chosen randomly).

Create an empty path.

Mark the initial city as visited.

While there are unvisited cities:

Find the nearest unvisited city from the current city.

Add this city to the path.

Mark this city as visited.

Update the current city to the newly visited city.

Add the starting city to the end of the path to complete the cycle.

Return the generated path.

Main Function (main):

Initialize coordinates and paths.

Print the given coordinates.

Apply Nearest Neighbor algorithm to find the path.

Calculate the total distance of the generated path.

Print the generated path and its total distance.

Plot the generated path.

3.3.2 Optimization - Lin-Kernighan heuristic

Algorithm:

Main Algorithm (linkernighan):

Start with an initial path, which can be generated randomly or by any other means.

Begin the loop for optimization:

Initialize a flag improved as True.

While improved is True:

Set improved to False.

Iterate through the path:

Consider all possible swaps of subsequences in the path and check if the resulting path is better than the current path.

If an improvement is found, update the path and set improved to True.

Repeat until no more improvements can be made.

Return the optimized shortest path.

Main Function (main):

Initialize coordinates and paths.

Print the given coordinates.

Generate an initial random path.

Calculate the initial total distance of the path.

Apply Lin-Kernighan algorithm to find the shortest path.

Calculate the shortest total distance of the optimized path.

Print the shortest path and its total distance.

Plot the optimized shortest path.

3.4 Data-Flow Diagram

One of the key components of our project design is the Data Flow Diagram (DFD). The DFD provides a visual representation of the flow of data within our system. It helps us understand how the system interacts with external entities and how data moves through it.

3.4.1 Level-0

The Level 0 DFD, also known as a context diagram, gives a broad overview of the system. It illustrates how the system interacts with external entities.

- System: The Level 0 Data Flow Diagram (DFD) of the Tourism Planner system includes three key components: admin side, user side, and the central server. The server acts as the central hub, facilitating data flow between the admin and user sides and playing a crucial role in managing and storing information.
- Admin-User Interaction: Admins provide necessary data and updates to the server, ensuring the system's information remains accurate and up-to-date. Users access the system through the user interface, retrieving data from the server for travel planning purposes.
- Server Functionality: The server hosts the interactive map feature and manages data related to user contributions, such as adding enterprises or businesses. It enables collaboration, synchronization, and scalability, ensuring a seamless flow of information between admin and user sides while maintaining optimal system performance.

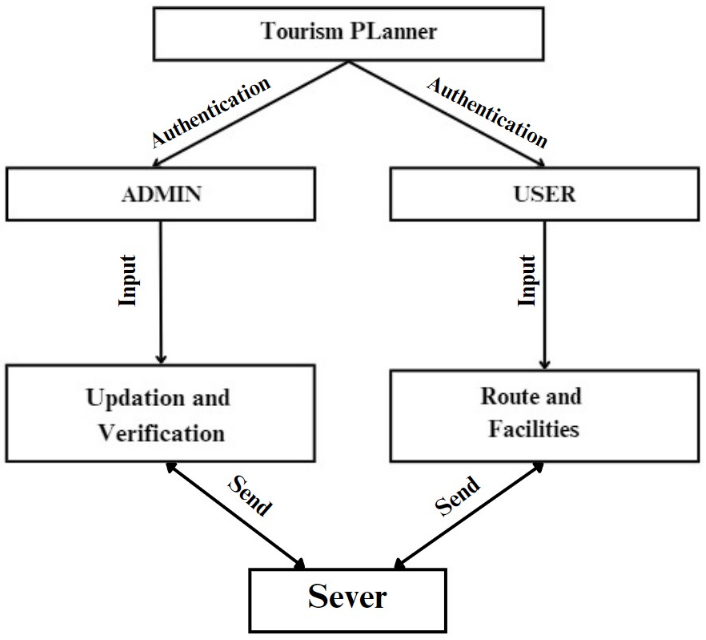


Figure 3.4: Data-Flow Diagram Level-0

This Level 0 DFD is a crucial part of our project design as it provides a high-level understanding of the system's processes and the flow of data. It aids in the clear communication of how the system is designed to work, making it an invaluable tool in our project design documentation.

3.4.2 Level-1

A Level 1 Data Flow Diagram (DFD) offers a detailed perspective of a system, breaking down major processes from the Level 0 DFD into sub-processes. Each subprocess is represented as a separate process, displaying associated data flows and data stores. This level provides a nuanced view, illustrating how data flows within the system and interacts with various entities. Essentially, it serves as an "exploded view" of the context diagram, offering insight into the main functions of the system. The choice of DFD level depends on the system's complexity and the desired level of detail, with higher levels providing a broad overview and lower levels delving into specific processes, data flows, and data stores. A combination of different DFD levels ensures a comprehensive understanding of the system

- User Side Interaction: Users can provide destinations they would like to visit through the system. The destination data is sent to the server for processing.
- Optimal Path Calculation: The server processes the user-provided destination

data to generate an optimal path. The optimal path is then provided to the user.

- Entrepreneur Interaction: If the user is an entrepreneur, there is an option for them to add their business to the Tourism Planner.
- Entrepreneur Data Submission: Entrepreneurs can provide the necessary data required for adding their business to the Tourism Planner.
- Admin's Role: The role of the Admin is to verify the data submitted by users and entrepreneurs.
- Data Verification and Update: Admin verifies the data that needs verification. After verification, the Admin updates the verified data to the server.
- Server Functionality: The server acts as a central hub for processing user-provided destinations and managing added businesses.

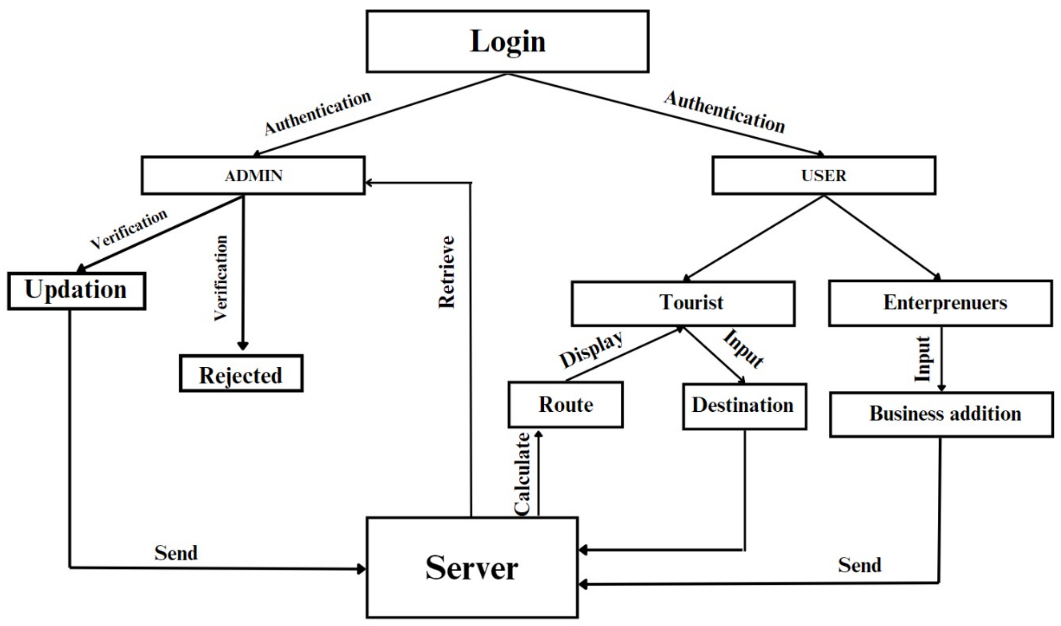


Figure 3.5: Data-Flow Diagram Level-1

3.4.3 Level-2

- Modules in Tourism Planner: The Tourism Planner comprises three main modules: user, admin, and server.
- Authentication Process: Authentication is used to verify whether a person is a user or an admin.

- User Module: Users can utilize the Tourism Planner for tour planning. To plan a tour, users provide their starting destination and places they would like to visit.
- Data Flow to Server: User-provided tour data is sent to the server for processing.
- Optimal Path Calculation: The server processes the user's data and returns an optimal path to the user.
- Entrepreneur Interaction: Users who want to add a business to the Tourism Planner need to submit necessary data.
- Business Data Submission: Entrepreneurs provide required data along with specifying the category for their business.
- Data Flow to Admin for Verification: The submitted data is sent to the admin side through the server for verification.
- Admin's Responsibilities: Admin verifies the authenticity of the submitted businesses. Admin is also responsible for adding new tourist-friendly places and making changes to the map.
- Centralized Server Role: The server acts as a centralized component for processing user requests, verifying business data, and managing additions to the map.

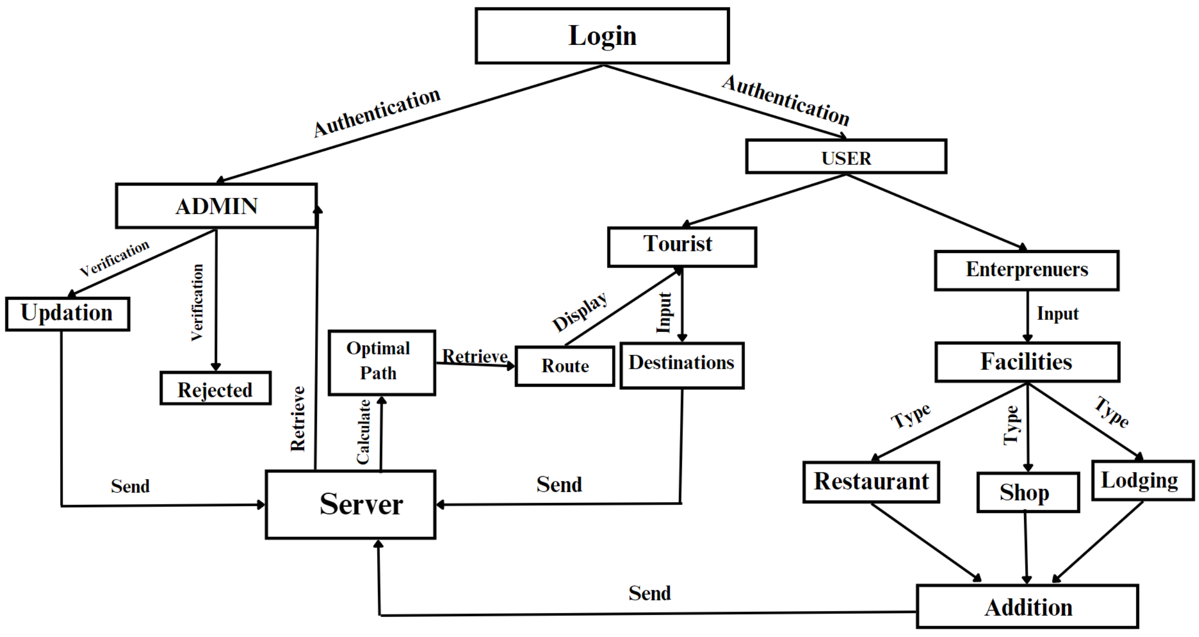


Figure 3.6: Data-Flow Diagram Level-2

3.5 Database Design

The foundation of our forum system lies in a well-thought-out database design, visualized through an Entity-Relationship (ER) Diagram. This diagram serves as a crucial guide, offering a comprehensive overview of the data entities and relationships within our system. It acts as a blueprint, illuminating the structure of our database and facilitating a clear understanding of data flow and dependencies.

- Entities: These represent the fundamental building blocks of our system, each playing a distinctive role
- Relationships: These connections define the interplay between entities, elucidating the dynamics of our system

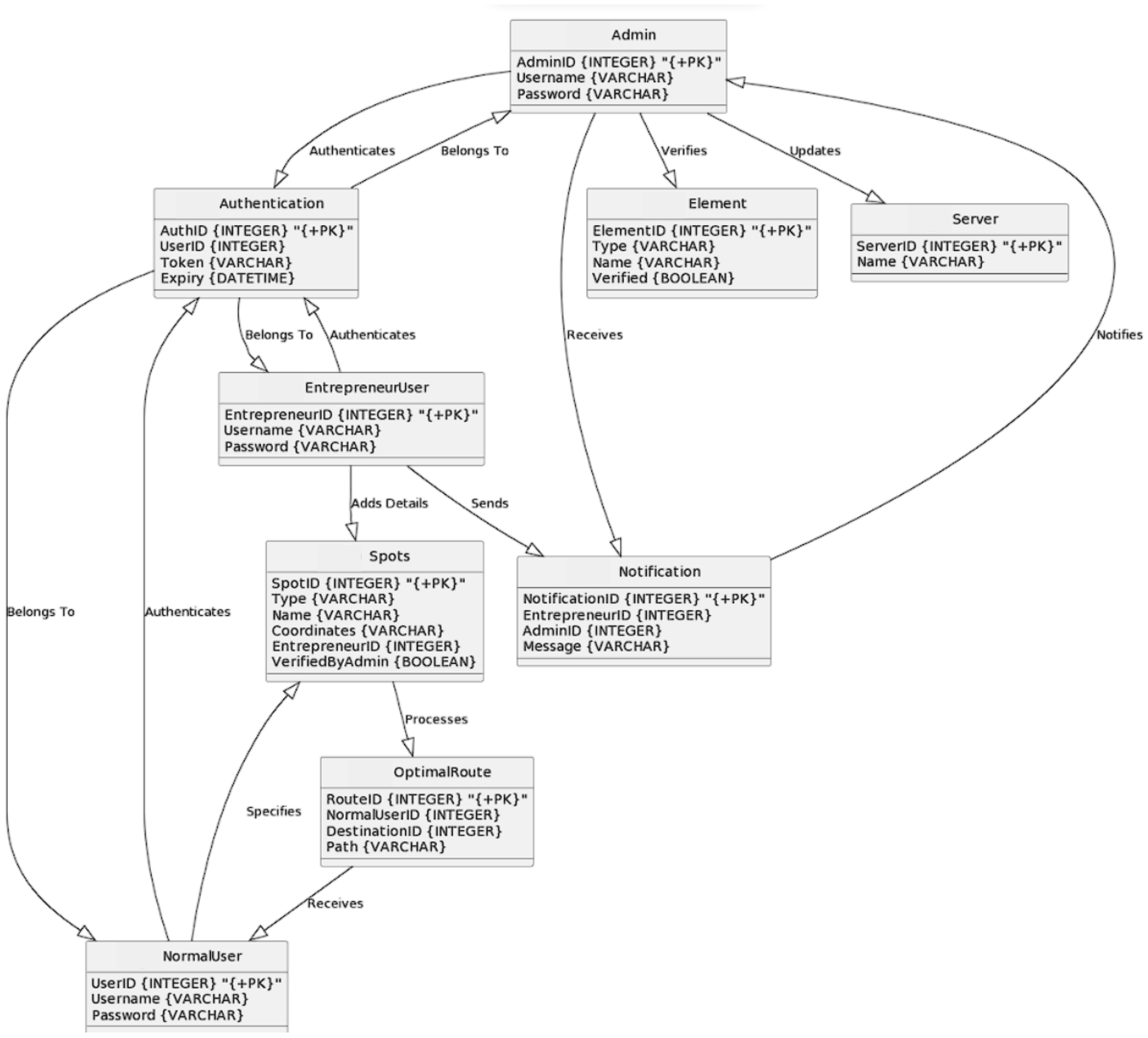


Figure 3.7: ER Diagram

3.6 Hardware & Software Requirements

Hardware:

- Smartphone(with android 8 or more)
- Laptop(processor i5 or greater)
- RAM 8gb or more
- Hard-disk 160gb or above

Software:

- Development tools such as Visual Studio Code (VSCode) for coding and project management.
- Fronted framework - Flutter, ReactJS
- Backend development framework used for web and application development - NodeJS and Hive
- Database Management System used for data storing and retrieval
  First choice - MongoDB
  Second choice - Firebase
- Web hosting platform provided through the GitHub Student Pack or from College server.

3.7 Work Schedule

- **October to December (2023):** Get Familiarized with Google Maps API features. Analyzing various approximation and optimization algorithms related to tsp. Review and understand the best algorithm for implementation. Begin designing the project, including conceptualization and initial UI/UX planning.
- **January to April (2024):** Get Familiarized with ReactJs, Flutter and NodeJs. In-depth learning of Python. Commence the project development phase. Work on user interface development and Implementation of data structure and algorithm visualization. Address any technical challenges that arise during development. Then we enter the debugging and testing phase.

4. Result & Conclusions

4.1 Design Phase Outcomes

The initial phases of the Tourism Planner have been successfully executed. Team formation and project idea making set the groundwork for our exploration into the project domain. Significant progress was made in conceptualizing the Tourism Planner through the creation of Data Flow Diagrams (DFDs) and System Architecture. These visual representations have provided a comprehensive view of the system's data flow and processes.The literature review served as a foundation, providing valuable insights and necessary algorithms for our project.

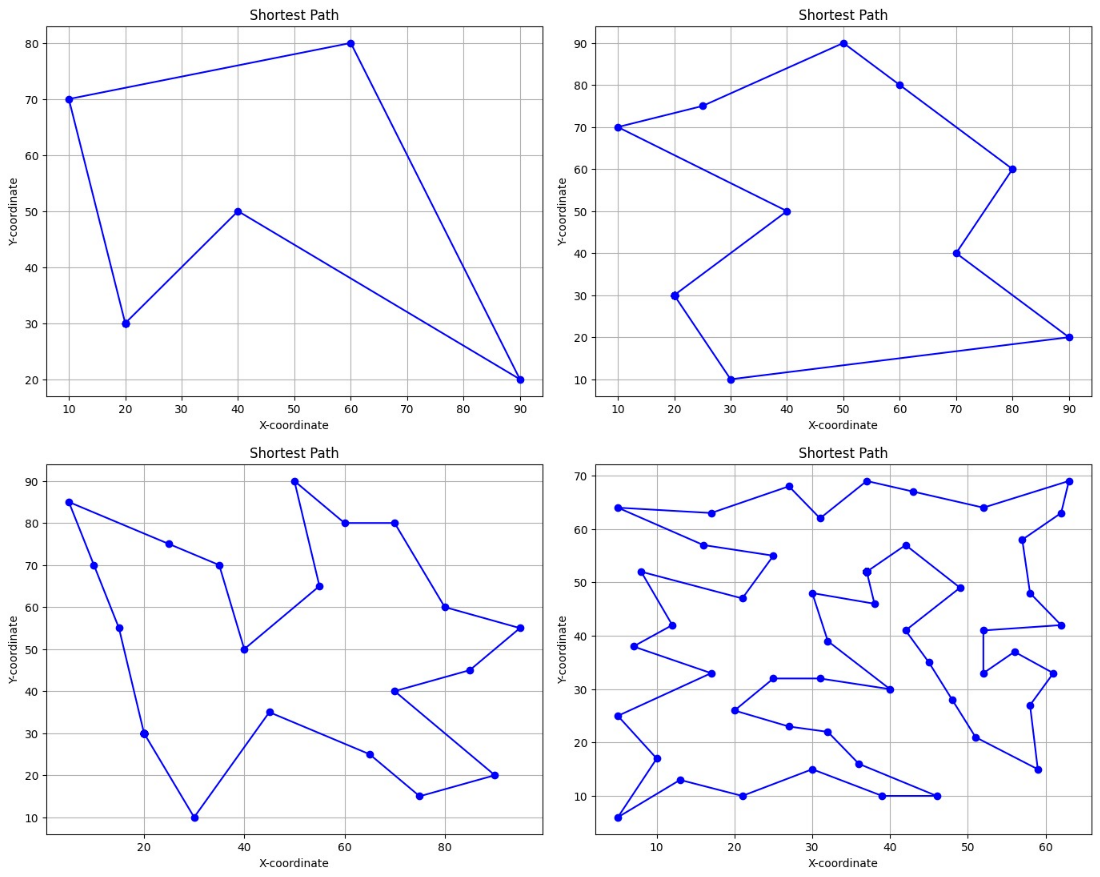


Figure 4.1: Lin-kernighan Trial Execution

The figures shows the outcome of the analysis of various algorithms and core part of our design phase. The figure 4.1 shows the result of Lin Kernighan algorithm for number of coordinates 5,10,20 and 50. In order to reduce the execution time when the number of coordinates increased the nearest neighbour algorithm is implemented. The result for number of coordinates 5,10,20 and 50 is given in figure 4.2.

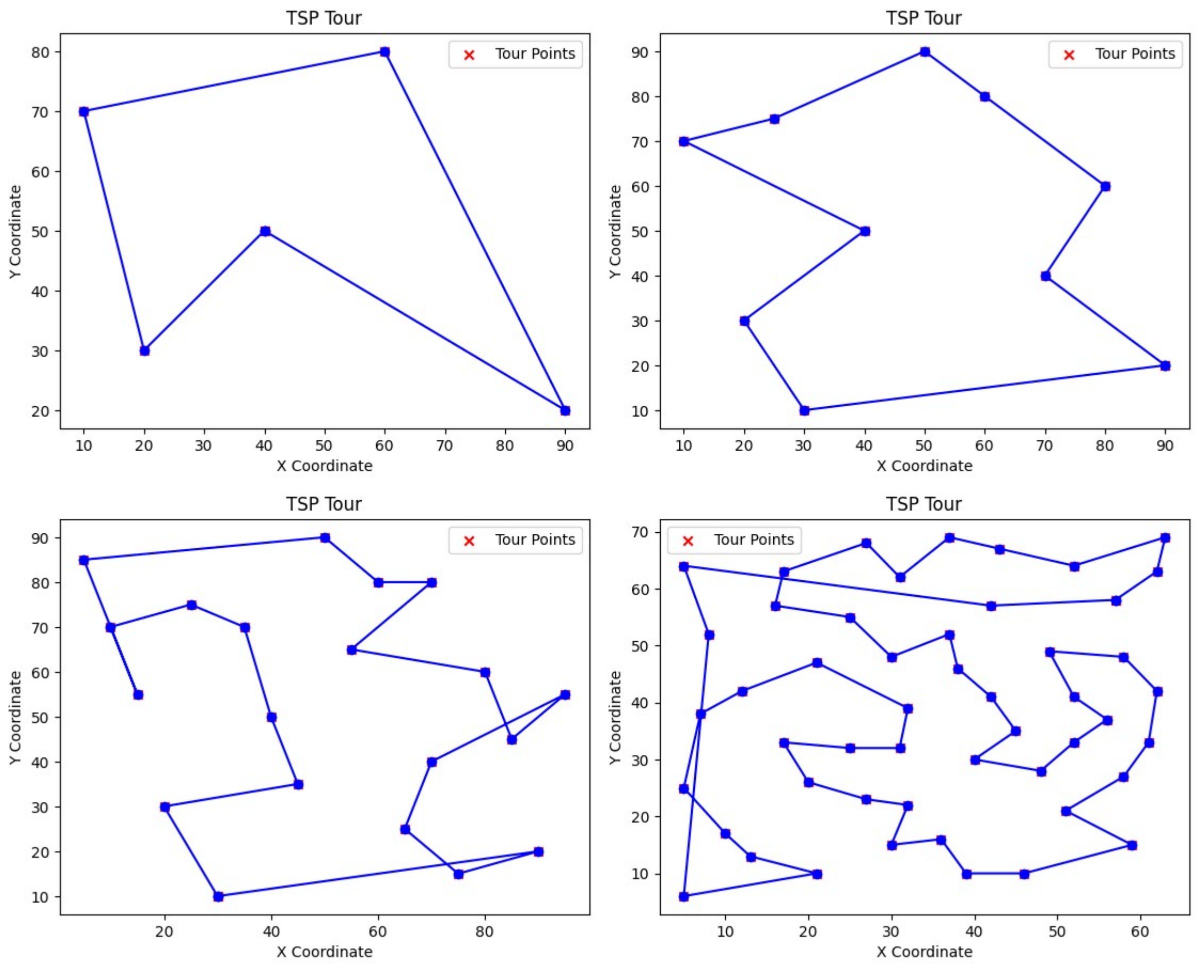


Figure 4.2: Nearest Neighbout Trial Execution

4.2 Conclusion

The successful completion of the design phase a crucial milestone in the development of Tourism planner. As we iterated through various algorithms, we learned the working and how it could be implemented in our application and how it should be implemented to produce the best results. In conclusion Lin Kernighan is suitable for provide route when number of destination around 10 and Nearest neighbour when there are more than 10 Destination.

References

[1] Federico Greco, “IntechOpen, Travelling Salesman Problem” 2008 DOI:10.5772/66, ISBN:978-953-7619-10-7

[2] Heinrich Braun “On solving travelling salesman problems by genetic algorithms” 01 January 2005 Institut fiir Logik, Komplexitht und Deduktionssysteme, Universi t Karlsruhe Posffach 6980, D 7500 Karlsruhe, Deutschland, e-maih braun@ira.uka.de

[3] Aleksandar Pejic; Szilveszter Pletl; Bojan Pejic “An expert system for tourists using Google Maps API” 2009 7th International Symposium on Intelligent Systems and Informatics DOI: 10.1109/SISY.2009.5291141