VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
INTERNATIONAL UNIVERSITY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

APPLYING GENETIC ALGORITHM FOR SOLVING MULTI-DEPOT VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

Submitted in partial fulfillment of the requirements for the Degree of Bachelor of Engineering in Logistics and Supply Chain Management

Student: Tran Le Phu
ID: IELSIU19237
Thesis Advisor: PhD. Nguyen Van Chung

Ho Chi Minh City, Vietnam
August/2023

APPLYING GENETIC ALGORITHM FOR SOLVING MULTI-DEPOT VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

By

Tran Le Phu

Submitted in partial fulfillment of the requirements for the Degree of
Bachelor of Engineering in Logistics and Supply Chain Management
International University, Ho Chi Minh City
August/2023

Signature of Student: _____

Tran Le Phu

Certified by _____

PhD. Nguyen Van Chung
Thesis Advisor

Approved by _____

Dr. Nguyen Van Hop
Dean of IEM school

ABSTRACT

The optimization of the multi-depot Vehicle Routing Problem (VRP) with time windows is a complex issue within the logistics and transportation field that traditional optimization methods struggle to solve due to its complexity. In contrast, Genetic Algorithm (GA) is a metaheuristic optimization method that has shown promising results in addressing various optimization issues, including the VRP. This research aims to implement GA for solving the multi-depot VRP with time windows and assess its performance regarding solution quality and computation time. The problem is transformed into a set of chromosomes, and genetic operators are employed to generate novel solutions. The fitness of each solution is assessed using multiple objective functions, including the total distance traveled, the number of vehicles utilized, and the total delivery time. The findings indicate that GA can identify nearly optimal solutions in a reasonable amount of time and handle dynamic and uncertain environments. Future studies may explore the potential of integrating machine learning algorithms and other metaheuristic optimization techniques to enhance the performance of GA in solving the multi-depot VRP with time windows.

Keywords: Vehicle Routing Problem, Genetic Algorithm, metaheuristic optimization, time windows, multi-depot.

ACKNOWLEDGEMENT

Firstly, I would like to sincerely express my gratitude to my advisor, PhD. Nguyen Van Chung for giving me such great advice for deciding the topic for thesis, as well as great approaches for thesis related problems. Mr. Chung also shared with me great thesis related documents. And finally, for leading me to the right path, to the success of the thesis. Secondly, I would like to give my thanks to my friends for great discussions to find the best approach for all problems, both related and non – related, and for sharing useful information.

Finally, thanks to GitHub, a well-known forum for programmers, for sharing documents and giving answers related to the searched problems. When I want to search about programming related problems, GitHub is usually the main reference. Also, I would like to thank VRP-REP for contributing such a great and free data source for my thesis topic.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Definition
VRP	Vehicle Routing Problem
GA	Genetic Algorithm
MDVRPTW	Multi-depot Vehicle Routing Problem with time windows
TSP	Traveling Salesman Problem
ML	Machine Learning
AI	Artificial Intelligence
SA	Simulated Annealing
TS	Tabu Search
MDVRP	Multi-depot Vehicle Routing Problem
DEAP	Distributed Evolutionary Algorithm in Python
PyGMO	The Python Parallel Global Multi-Objective Optimizer
GPS	Global Positioning System
IDs	Identifications

Chapter 1. INTRODUCTION

1.1. Background of the study

Logistics is a vital component of supply chain management, which is an essential feature of modern business operations. It entails the efficient flow of commodities, services, and related information from their site of origin to their point of consumption, in accordance with customer requirements. The concept of logistics arose from military uses and rose to prominence during the First and Second World Wars. Today, logistics has become a crucial element in most businesses, allowing them to run their operations successfully and efficiently.

The term "logistics" has evolved through time to better describe how the industry has developed. Logistics is described as "the branch of military science relating to procuring, maintaining and transporting material, personnel and facilities" in the Oxford English Dictionary. However, the New Oxford American Dictionary defines logistics as "the detailed coordination of a complex operation involving many people, facilities, or supplies," and the Oxford Dictionary online defines it as "the detailed organization and implementation of a complex operation." Because of this, logistics is seen as an engineering discipline that focuses on developing "people systems" rather than "machine systems."

In military logistics, it is concerned with maintaining the supply chains that allow armies to transport their troops as well as obtain food, ammunition, weaponry, and replacement parts. Civil logistics, on the other hand, deals with the acquisition, delivery, and storage of raw materials, semi-finished goods, and finished goods in non-military settings. Additionally, logistics are crucial in the service sector, where businesses that provide services like garbage collection, mail delivery, public utilities, and after-sale services, must consider logistical concerns.

By understanding the concept and purpose of logistics, it helps us comprehend the importance of transportation in both logistics and supply chain management. Overall, this can increase the efficiency and efficacy of logistics and supply chain management

operations by enabling more strategic decisions to be made about transportation networks. Businesses may also use technology to automate and improve their supply chain management and logistics procedures, which will boost output, reduce costs, and improve customer satisfaction.

1.2. Problem Statement – The Need for Study

The logistics industry plays a critical role in the global economy, and logistics management is a key factor in the success of companies. Without efficient logistics, businesses are unable to satisfy client demands and expectations, which results in lost sales and money. While some businesses have successfully implemented logistics management methods, many others have had difficulty coming up with a workable plan. Because of this, these businesses have wasted a lot of time and money, both installing logistics systems and making up for errors that were made along the way. Even if they make an attempt to optimize their truck routing, delays in delivery might still occur due to problems like traffic jams, road construction, and unanticipated occurrences like parades or accidents.

There are two basic reasons for the difficulties in logistics. First off, flexible and universal solutions that can completely address real-world difficulties are quite rare, even though there have been many research projects focused at tackling specific situations of logistical challenges, notably those connected to transportation. This is because logistical restrictions cannot be predicted or estimated using static or mathematical techniques since they are no longer predictable variables. The need for personal automobiles rises along with the city's population and occupational diversity, making mobility increasingly difficult and flexible. Additionally, the city's traffic arrangement is unsuited for large deliveries since the government prohibits the admission of huge vehicles.

Second, the growth of internet commerce is a result of the ongoing development of technology. For instance, corporations have switched from traditional commerce and service models to e-commerce ones because of the government's Industry 4.0 strategy. Transportation reaction times must be as swift and flexible as possible, and decisions must be made quickly. Finally, two goals are critical in logistics: the creation of a genetic

algorithm for realistic situations and the use of information technology such as real-time monitoring and traffic detection.

One of the typical examples for the challenges logistics management today is the Multi-Depot Vehicle Routing Problem with Time Windows (MDVRPTW). It is a complex optimization problem that involves finding the most efficient way to transport goods or services from multiple depots to a set of customers while considering various constraints such as vehicle capacity, time windows, and depot locations.

While there are existing optimization techniques available for solving VRP, they have limitations that prevent them from providing optimal solutions for large-scale problems or finding solutions within a reasonable time frame of the MDVRPTW. As a result, these limitations can have significant implications for logistics management, such as longer transportation times, higher transportation costs, and lower customer satisfaction.

This thesis aims to examine this problem by exploring the application of GA to solve the MDVRPTW in logistics field and evaluating its effectiveness compared to other optimization techniques. The study aims to identify the strengths and limitations of GA in solving the MDVRPTW and to propose new optimization techniques that can be used by logistics department of companies.

1.3. The Design Project Objectives and Requirements:

1.1. Expected benefit from this study

The results of this study will benefit logistics managers, policymakers, and researchers who are involved in the transportation industry. Specifically, logistics managers will be able to utilize the proposed approach to optimize their vehicle routing plans, thereby enhancing their transportation efficiency and reducing transportation costs. Policymakers will be able to utilize the findings to develop policies that promote more efficient and cost-effective transportation systems. Furthermore, researchers in the field of logistics management will benefit from the new insights and practical implications presented in this study, which can be used to develop further research in this area. Ultimately, the proposed approach has the potential to benefit businesses and organizations by improving

their logistics management practices, reducing costs, and enhancing their competitive advantage in the marketplace.

1.2. Expected output and application

In this project, we desired to obtain the expected outputs:

- Optimal or near-optimal routes for each vehicle
- A list that indicates the total duration of each vehicle
- A table that shows the total distance traveled by each vehicle

In short, this study's major emphasis can be applied to solve the multi-depot Vehicle Routing Problem with Time Windows by generating optimized routes for each vehicle, minimizing travel distance, and satisfying time window constraints. This has practical applications in logistics, transportation, and emergency services by reducing costs, time, and fuel consumption while ensuring customer satisfaction.

1.3. Design Requirements

- Optimal solution: The system should be able to generate optimal or near-optimal solutions for the multi-depot VRP with time windows.
- Constraints satisfaction: The system should ensure that the generated solutions satisfy the time window constraints, vehicle capacity limits, and other relevant constraints.
- Efficiency: The system should be able to find solutions within a reasonable time frame to ensure operational efficiency.

1.4. Scope and Limitation

The scope of this project is to develop a model that can solve the multi-depot Vehicle Routing Problem with Time Windows (MDVRPTW) by using genetic algorithm-based approaches. The system will consider a finite number of vehicles, depots, and customers, and their associated time window constraints. The goal of the study is to generate optimal or near-optimal solutions that minimize the total travel duration of the vehicles while ensuring that all constraints are satisfied. The system will also have an intuitive and user-

friendly interface that allows users to input problem parameters and visualize the solutions.

Despite the scope of this project, there are several limitations that may affect the model's performance. Firstly, the system may face computational and memory limitations when dealing with large-scale problems. Secondly, the system may not be able to handle problems with a high degree of uncertainty and noise in the input data, such as unpredictable demand fluctuations or road closures. Additionally, the accuracy of the solutions generated by the system may be affected by the quality and completeness of the input data. The system may also be limited in its ability to account for all operational and business constraints, such as driver availability, vehicle maintenance schedules, and customer preferences. Finally, the system's performance may be affected by hardware and software limitations and may require significant computational resources and time to generate optimal solutions for large-scale problems, which may not be feasible in real-time applications.

Chapter 2. DESIGN CONCEPTS CONSIDERATION

2.1. Overview

The Vehicle Routing Problem (VRP) is a well-known optimization problem in the subject of operations research. Its goal is to identify the best or nearly best scheduling and routing of vehicles to serve a set of clients while minimizing total transportation costs. When further restrictions are added to the situation, such time boundaries and different depots, the issue becomes more complicated. Toth and Vigo (2002) [1] provides an outline of the VRP and numerous methods to solutions.

"Machine learning" is a branch of artificial intelligence that focuses on developing statistical models and algorithms that enable computers to infer new information from their prior performance without needing to be explicitly programmed. The application of machine learning approaches to resolve challenging optimization issues like the VRP and its expansions has been researched and summarized by Czuba and Pierzchala (2021) [2] in recent years.

2.2. Literature Review

The Multi-Depot Vehicle Routing Problem with Time Windows (MDVRPTW) is an extension of the Vehicle Routing Problem (VRP) that deals with determining the best or nearly best routing and scheduling of a fleet of vehicles from several depots to service a group of clients with certain time window restrictions while minimizing the overall transportation expenses. The MDVRPTW is a real-world and challenging issue that appears in a number of applications, such as logistics, freight transportation, and public transit.

The MDVRPTW, according to Bae and Moon (2016) [3], is a problem that involves determining the best routes for a fleet of vehicles to deliver goods to customers across multiple locations with time window restrictions, taking into account the existence of multiple depots at different locations, while minimizing the overall distance traveled, the number of vehicles used, and the time taken to complete the routes, subject to various constraints. Time periods, various depots, which serve as the beginning and terminating

destinations for the vehicles, and capacity restrictions, which set a maximum weight limit on the amount of cargo that may be delivered by a vehicle are some of these restrictions. The MDVRPTW is an extremely difficult problem, as it builds on the already complex VRP by adding time window constraints and multiple depots. The inclusion of time windows is crucial for resolving vehicle routing issues since it gives the situation more reality. Customers frequently have certain time slots in which they anticipate getting deliveries or services in real-world settings. Ignoring these limitations may lead to dissatisfied clients, resource waste, and missed business opportunities.

Moreover, time windows can have a substantial impact on the algorithm's effectiveness and solution quality. They place extra restrictions on the routes and force the algorithm to discover ideal or almost ideal solutions that take into account the clients' time limits. As a result, taking time frames into account can result in more workable and efficient solutions, which can significantly affect the effectiveness and profitability of transportation and logistics organizations. Due to its high complexity, many heuristic approaches have been proposed to address the problem.

Tan et al. (2001) [4], introduced three mainly third-generation artificial intelligent (AI) algorithms, namely simulated annealing (SA), Tabu search (TS) and genetic algorithm (GA) to solve the vehicle routing problem with time windows (VRPTW). This paper applied each of the heuristics developed to Solomon’s 56 VRPTW 100-customers instances, observed the result of these three heuristics to the dataset and had an insight about the pros and cons of each method. Also, we developed our GA for solving MDVRP with time frames based primarily on this research.

Chapter 3. SYSTEM DESIGN

3.1. Approaches Comparison and Selection

3.1.1. Simulated Annealing (SA)

Simulated annealing (SA) is a stochastic relaxation technique that has its roots in statistical mechanics. For the purpose of resolving combinatorial optimization issues, SA is an effective metaheuristic algorithm. It is based on the metallurgical annealing process, in which a material is heated and then gradually cooled to get the required structure. Similar to this, the simulated annealing technique creates potential solutions and iteratively seeks out a better one by perturbing the present solution at random and assessing the objective function. A temperature parameter in the algorithm regulates the likelihood of accepting an inferior result. To avoid local optima, the algorithm first accepts a lot of poorer answers. The algorithm grows increasingly selective as the temperature drops and eventually converges to the global optimum.

Simulated annealing has been successfully used to a variety of optimization issues, including the traveling salesman problem, the knapsack problem, and the vehicle routing problem. It is particularly useful for problems with a large search space, where traditional optimization techniques may get stuck in local optima.

For the MDVRP with backhauls, Karimi and Javad (2017) [5] provided a simulated annealing approach that takes into account going back to the depot after a delivery to pick up extra products. The specific "green" vehicle routing problem was solved by Karagül et al. (2018) [6] using simulated annealing. The MDVRPTW was subjected to multi-objective simulated annealing by Wei et al. (2019) [7], which took into account numerous competing objectives including reducing the overall journey time and the number of cars employed. A new simulated annealing approach was put up by Amine (2019) [8] for the capacitated vehicle routing problem, a subset of the VRP.

Simulated annealing has also been used in combination with other metaheuristic algorithms to solve the MDVRPTW. Kancharla and Ramadurai (2020) [9] developed an effective simulated annealing algorithm that uses a local search procedure to enhance the

solution quality for the MDVRPTW.

3.1.2. Tabu Search

Another well-known metaheuristic technique is Tabu Search, which has been used to address issues with combinatorial optimization, such as the multi-depot vehicle routing problem with time windows (MDVRPTW).

While solving the MDVRPTW with backhauls, Hu (2020) [10] explores a combined optimization approach utilizing tabu search to solve the multi-depot vehicle routing problem. It was demonstrated that the algorithm can generate excellent results in a reasonable amount of time. The MDVRPTW with cross-docking, which entails the transshipment of commodities from one vehicle to another at a common point, was solved by Zhang and Niu (2009) [11] using Tabu Search. Their findings demonstrated that Tabu Search can efficiently resolve the issue with top-notch solutions.

Tabu Search has also been integrated with other metaheuristic algorithms to improve its performance in solving the MDVRPTW. A hybrid approach that combines Tabu Search with the ant colony optimization technique, for instance, was proposed by Li et al. (2020) [12] to solve the MDVRPTW. In terms of computing time and solution quality, it has been demonstrated that the hybrid method performs better than the standalone Tabu Search algorithm.

3.1.3. Genetic Algorithm

The genetic algorithm is a metaheuristic approach inspired by natural processes like inheritance, mutation, selection, and crossover. It is a subset of a larger class of evolutionary algorithms that employ these methods to identify the best answers to a range of issues. The process of creating a fitness function, a representation, and genetic operators that are unique to the problem being optimized is part of the process of employing a genetic algorithm to solve problems. In this paper, instead of clustering and allocating clients to each depot separately, we suggested a genetic approach for solving the MDVRP by taking into account all depots concurrently. Our genetic algorithm can be used to solve a variety of optimization issues; it is not just applicable to the MDVRP.

Ochelska-Mierzejewska et al. (2021) [13] conducted a comprehensive investigation of selected genetic algorithms for solving the VRP, including the MDVRPTW. Using a set of benchmark examples, the study assessed the effectiveness of various GAs, including the regular GA, the hybrid GA, and the adaptive GA. According to the results, the adaptive GA surpassed other algorithms in terms of the accuracy of the solutions and the efficiency of computing.

In order to create new solutions, Rybickova et al. (2015) [14] presented a GA-based method for resolving the MDVRP. This method uses crossover and mutation operators. On benchmark examples, the strategy was evaluated and contrasted with other methods. The outcomes demonstrated that the GA-based strategy performed better in terms of solution quality than other approaches.

Table 3.1: Advantages and disadvantages of potential approaches

Algorithm	Advantages	Disadvantages
Simulated Annealing (SA)	Easy to implement Able to escape local optima Can handle large solution spaces	Slow convergence rate Able to converge to suboptimal solution Heavily dependent on initial parameters
Tabu Search	Able to escape local optima Can handle large solution spaces Can incorporate additional constraints	Able to converge to suboptimal solution Heavily dependent on initial parameters Computationally expensive
Genetic Algorithm	Can handle large solution spaces Can incorporate additional constraints Can handle multiple	Computationally expensive Able to converge to suboptimal solution Requires careful tuning of parameters

	objectives Parallelizable	

Based on these advantages and disadvantages, it can be concluded that genetic algorithm is the suitable algorithm for solving the Multi-Depot Vehicle Routing Problem with Time Windows due to its ability to handle large solution spaces and incorporate additional constraints and solve multiple objective problems. Furthermore, with the availability of optimization libraries in Python, such as DEAP, it is relatively easy to implement and customize genetic algorithm models to fit specific problem requirements.

3.2. Introduction to Python

Python is a well-liked high-level programming language that is often employed in many different industries, including data analysis, machine learning, web development, and scientific computing. It was created by Guido van Rossum in the late 1980s and released in 1991. Python is recognized for its simplicity, readability, and ease of use. Its syntax is clear and concise, making it a great language for beginners.

In the context of solving the multi-depot VRP with time windows, Python can be used to develop and implement genetic algorithm-based optimization models that can help to find near-optimal solutions to the problem.

Python has several advantages that make it suitable for solving optimization problems. Firstly, it has a large and active user community that provides support and develops libraries, such as NumPy and Pandas, which can help in data processing and optimization tasks. Secondly, Python is open-source and free to use, making it a cost-effective solution for researchers and practitioners. Thirdly, Python is a versatile language that can be used for a wide range of tasks, from data analysis to web development.

In the context of solving the multi-depot VRP with time windows, Python can be used to develop a genetic algorithm-based optimization model that can find near-optimal solutions to the problem. The model can be implemented using a combination of Python's built-in data structures, such as lists and dictionaries, and third-party libraries, such as DEAP and PyGMO, which provide tools for implementing genetic algorithms.

By using Python to solve the multi-depot VRP with time windows, researchers and practitioners can benefit from the language's flexibility, ease of use, and powerful libraries. They can also easily modify the optimization model to incorporate new constraints or objectives and test its performance on different problem instances. Hence, with its ease of use, flexibility, and wide range of libraries, Python is a suitable programming language for this study working on solving optimization problems such as the MDVRPTW.

3.3. Solution Approach Description: Design description

3.3.1. Design Structure

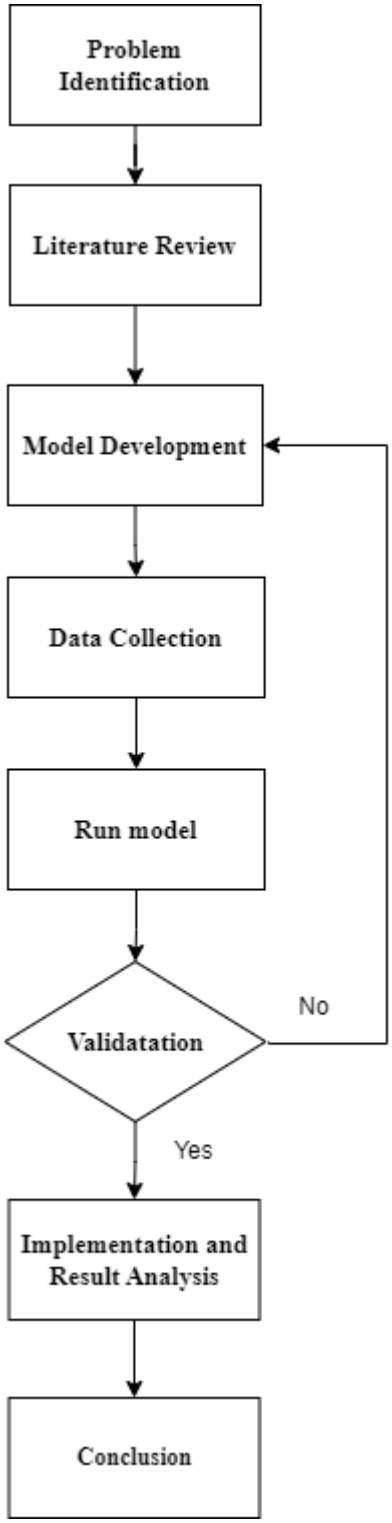

Figure 3.1: Flowchart of the thesis

- Step 1: Problem Identification

Investigate to find a suitable problem that is happening in logistics industry. Define the problem in the problem statement and explain the scope and limitations of the thesis.

- Step 2: Literature Review

Research the paper that related to the problems and review the potential approaches that can apply to solve the MDVRPTW.

- Step 3: Model Development

Define the mathematical model including objective function, constraints, parameters, and variables for the problem. Construct the framework of GA following the model and set the termination criteria for the GA.

- Step 4: Data Collection

Collect the necessary data, including the location of customers, their demands, and the depot locations. Additionally, the time windows and vehicle capacity constraints should be considered.

- Step 5: Run model

Run the model with sample dataset to test the performance and accuracy of the model.

- Step 6: Validation

If the result of the model is passed, go to collect data that has satisfied the requirements of the model. Otherwise, continue improving the model.

- Step 7: Implementation and Result Analysis

Implement the algorithm and test it on various problem instances to evaluate its effectiveness and efficiency.

The Results & analysis section will reveal the outputs of the algorithms and their significant changes, as well as their effectiveness.

- Step 8: Conclusion

Based on the Result and Analysis, draw conclusions on the effectiveness of the genetic algorithm for solving the MDVRPTW. Identify areas for further research and improvement.

3.3.2. Key advantages of the proposed alternatives

The proposed process for building a model to solve the MDVRPTW using a genetic algorithm offers several key advantages. Firstly, genetic algorithm is known for its ability to handle complex problems with large solution spaces. This makes it a suitable choice for solving the MDVRPTW, which is a highly complex optimization problem. Secondly, the proposed process involves using Python programming language, which is a popular and versatile language for scientific computing and data analysis. This ensures that the implementation of the genetic algorithm will be efficient and effective. Finally, the proposed process involves using a benchmark study to assess the performance of the genetic algorithm and make improvements where necessary. This ensures that the final solution will be highly optimized and effective in solving the MDVRPTW. Overall, the proposed process offers a robust and effective approach to solving the MDVRPTW using a genetic algorithm.

3.3.3. Framework of the Genetic Algorithm

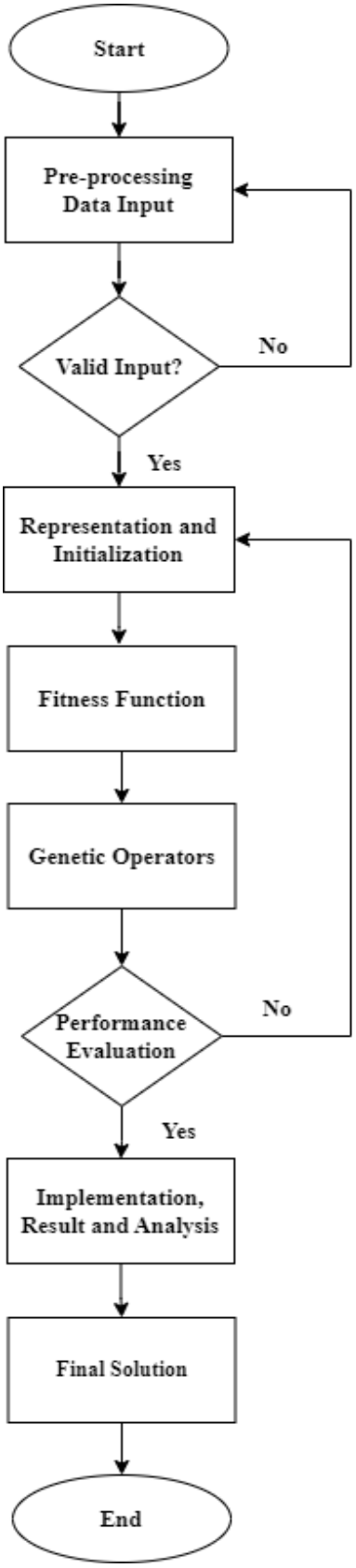

Figure 3.2: Genetic Algorithm Framework

- Pre-processing Data Input: Transform the collected data, including the location of customers, their demands, the depot locations, the time windows, and vehicle capacity constraints into the format of attribute, class, data type and size.
- Valid Input: Check the validation of the data input before implementing the model.
- Representation and Initialization: Determine how to represent the solution as a chromosome. One approach is to use a binary string to represent the order of the customers for each vehicle. Then, generating an initial population of chromosomes which can be done randomly.
- Fitness function: Define a fitness function that evaluates the quality of each chromosome. This function should consider the objective function (minimizing the total distance) and the constraints (time windows and vehicle capacity).
- Genetic operators: Implement the genetic operators of parent selection, crossover, and mutation to generate new offspring chromosomes from the parent chromosomes.
- Model Evaluation: Set the termination and parameter condition for the model. Get feedback for model and do improve it if necessary.
- Implementation and Result Analysis: Implement the algorithm and analyze the result including sensitivity analysis on different changes in data input to evaluate its effectiveness and efficiency.
- Final Solution: At the end of the calculation, the user can modify the routes and vehicle assignments. The modified route will be saved in a different variable, recalculated, and compared to the algorithm's best feasible result. Users will choose the best option in the end.
- Conclusion: Conclude on the effectiveness of the genetic algorithm for solving the MDVRPTW. Identify areas for further research and improvement.

3.3.4. Data collection

The input data required for the MDVRPTW problem includes the following information:

1. Customer locations: The location of each customer needs to be determined and represented as (x, y) coordinates. This information can be obtained through GPS or manual input.
   x: represents for the longitude
   y: represents for the latitude
2. Customer demands: The demand of each customer needs to be determined and represented in terms of units or weight. This information can be obtained from sales records or by directly communicating with the customers.
3. Depot locations: The location of each depot needs to be determined and represented as (x, y) coordinates. This information can be obtained through GPS or manual input.
   x: represents for the longitude
   y: represents for the latitude
4. Vehicle capacity: The maximum load capacity of each vehicle needs to be determined and represented in terms of units or weight. This information can be obtained from the vehicle specifications or by consulting with the manufacturer. In this problem, we use six vehicles with the same maximum load.
5. Time windows: The time windows for each customer need to be determined and represented as start and end times. This information can be obtained from the customer's availability or by consulting with the customer.

These are requirements for the sample data of our study. And Vidal et al. (2013) [16] used severe computational experiments for a large class of Vehicle Routing Problem with Time Windows. From this paper, they proposed datasets for those who were interested in solving MDVRPTW. Those sample datasets which were contributed on VRP – REP website fits our requirements of the research. So, for this study, we will use one of sample datasets to evaluate our model.

3.3.5. Pre-processing Data input

The pre-processing steps include:

1. Assigning unique IDs to each customer and depot to ensure that they can be easily referenced in the algorithm.
2. Converting the (x, y) coordinates of the customer and depot locations into a distance matrix using the Euclidean distance formula.
3. Creating a demand matrix that lists the demand for each customer.
4. Creating a time window matrix that lists the start and end times for each customer.
5. Creating a capacity matrix that lists the maximum load capacity of each vehicle.

Chapter 4. SOLUTION DEVELOPMENT

4.1. Initial Mathematical Model

4.1.1. Vehicle Routing Problem with Time Windows

Tan et al. (2001) [4] claim that the Vehicle Routing Problem with time windows (VRPTW) is an extension of normal VRPs, which involves a fleet of vehicle set off from a depot to serve a number of customers, at different geographic locations, with various demands and within specific time windows before returning to the depot. The objective of the problem is to find routes for the vehicles to serve all the customers at a minimal cost (in terms of travel distance, etc.) without violating capacity and travel times constraint of the vehicles and the time windows constraints set by the customers.

4.1.2. Multi-Depot Vehicle Routing Problem with time windows

From the previous model, this study develops a mathematical model for the MDVRPTW. Instead of minimizing cost, the objective of our problem is to find routes for the vehicles to serve all the customers at a minimal distance.

Sets:

- D: Set of depot nodes
- C: Set of customer nodes
- N: Set of all nodes, N = $D \cup C$ (both customers and depots node)
- V: Set of vehicles

Parameters:

- n: Number of depots
- v: Number of vehicles
- d_i: The demand of customer i
- s_i: The service duration of customer i
- e_i: The earliest start time at customer i
- l_i: The latest start time at customer i
- L: The maximum load of each vehicle
- T: The maximum duration that each vehicle can be used

- c_{ij}: The Euclidean distance between node i to node j
- t_{ij}: The time required to travel from node i to node j
- (a_i, b_i): The time window for customer i

Variables:

- x_{ijk} = 1 if vehicle k travels from node i to node j, 0 otherwise $\forall\, i, j \in N, k \in V$
- t_i: The arrival time at node i $\forall\, i \in C$
- w_i: The waiting time at node i $\forall\, i \in C$
- z_{ik}: The start service time of vehicle k at node i $\forall\, i \in N, k \in V$

Objective function:

Minimize:

$$\sum_{i \in N} \sum_{j \in N} \sum_{k \in V} c_{ij} \times x_{ijk}$$

Subject to:

Constraint (1): Ensuring the vehicle that leaves the customer is the same as the one that visits the customer

$$\sum_{i \in N} x_{ijk} = \sum_{i \in N} x_{jik} \;\; \forall\, j \in N, k \in V$$

Constraint (2): Each customer is assigned to a vehicle

$$\sum_{i \in N} \sum_{k \in V} x_{ijk} \;=\; 1 \; \forall\, j \in C$$

Constraint (3): Determining whether each customer is assigned to a depot or not

$$\sum_{i \in C} \sum_{j \in D} x_{ijk} \;\leq 1 \; \forall\, k \in V$$

Constraint (4): The demand of each vehicle cannot exceed its maximum load

$$\sum_{i \in C} \sum_{j \in N} x_{ijk} \times d_i \;\leq\; L \; \forall\, k \in V$$

Constraint (5): The time used by each vehicle cannot exceed its maximum duration

$$\sum_{i \in N} \sum_{j \in N, i \neq j} x_{ijk} \times (t_{ij} + w_i + s_i) \leq\; T \; \forall\, k \in V$$

Constraint (6): The time including arrival time, delivery time, waiting time and service time at node i cannot exceed the arrival time at node j

$$\sum_{k \in V} \sum_{i \in N, i \neq j} x_{ijk} \times (t_i + t_{ij} + w_i + s_i) \leq t_j \; \forall \, j \in C$$

Constraint (7): The vehicle must arrive at the customer after the specified start time window of customer i

$$a_i \leq e_i + s_i \; \forall \, i \in C$$

Constraint (8): The vehicle must arrive at the customer before the specified end time window of customer i

$$b_i \geq l_i + s_i \; \forall \, i \in C$$

The objective function represents the total distance traveled by all the vehicles, while the constraints ensure that each customer is visited exactly once, each vehicle visits one customer at a time, and the capacity and time constraints of each vehicle are not violated. The time window constraints ensure that the vehicle arrives at each customer within the specified time window.

4.2. Genetic Algorithm

4.2.1. Representation and Initialization

For the genetic algorithm, generate a random starting population of chromosomes. To develop the initial individuals, which stand in for prospective answers, a step-by-step procedure is used. To prepare for vehicle-route assignments, the function begins by adding a vehicle to each depot. After that, a consumer is arbitrarily chosen by the function and momentarily added to a vehicle. As long as the total needs of all customers do not exceed the vehicle's maximum load, this process continues, adding more consumers to the vehicle. Additionally, the function makes sure that each customer's time frame requirements are satisfied while ensuring that the overall length of each route stays within the permitted range. Once these requirements have been met, the function changes the temporary load variable, adds a new car to the depot, and places the client inside it. It ultimately leaves the temporary array copy when more customers are added. This iterative technique enables the development of a variety of starting people while taking the

problem's needed load capacity and time restrictions into account.

4.2.2. Fitness function

The fitness function is created to evaluate how well a solution performs in terms of total duration and total distance traveled. An individual chromosome that represents a potential solution is fed into the fitness function. It then examines each vehicle's path along the chromosome to determine the overall distance and time associated with the chosen solution. Additionally, according to the mathematical model, the function includes penalty terms to account for violations of particular requirements.

The first constraint focuses on preventing discrepancies by assuring that every vehicle departing a client is the same vehicle that visits that customer. The second constraint deals with allocating each consumer to a certain vehicle, ensuring that no customer is visited more than once. In order to prevent unassigned customers, the third constraint decides whether or not every client is assigned to a depot.

The fourth constraint places a maximum load cap on each vehicle to avoid overloading. The function checks if the total demand of customers within a route exceeds the vehicle's capacity and imposes a fine if it does. The fifth constraint addresses the maximum time of each vehicle's path. If the overall time exceeds the allowable limit, a penalty is imposed. Finally, the fitness function contains three additional constraints related to time gaps between successive consumers. These restrictions make sure that every customer's arrival time, which includes service time, fits inside their assigned time periods. The appropriate sanctions are added if any violations are found.

The fitness value produced by the function indicates the entire performance of the solution, including the computed distances, durations, and penalties. The objective is to reduce fitness value since a lower fitness value suggests a more ideal and workable solution.

4.2.3. Parent Selection

Selection is a process in which individual strings produce a new population for the next generation depending on their fitness function values. In this research, we chose based on the Tournament. In the implemented tournament selection strategy, a fixed number of

individuals, known as the tournament size, are randomly selected from the population. These individuals compete, and the one with the best fitness value is selected as a parent. The selection process incorporates a probabilistic element based on a threshold parameter, which determines the probability of selecting the best individual in the tournament. This approach balances the preservation of fitter individuals while allowing for the exploration of diverse solutions. By striking this balance, tournament selection promotes the discovery of optimal solutions by favoring individuals with superior fitness values while maintaining genetic diversity within the population.

4.2.4. Crossover

For crossover, we use best cost route crossover, a route crossover method developed for vehicle routing with time window followed by Ombuki and Hanshar [15]. Since this approach is designed for VRPTW with only one depot, we only considered choosing one random depot when crossover method is called.

+ Randomly select p1, p2 from parent selection

+ Randomly select a depot

+ Randomly select a route from each parent, r1 from p1, r2 from p2

+ Remove all customers of r1 from p2, and r2 from p1

+ The next stage is to reconstitute the individual by altering the vehicle sequence once the consumers have been removed. The omitted routes are reinstated in their corresponding parents. The insertion method makes sure that the withdrawn clients are added to the progeny while still respecting the vehicles' capacity and time restrictions.

+ Finally, the best cost crossover operation produced the newly generated persons. These kids incorporate alterations to possibly increase the population's overall fitness while still inheriting the genetic makeup of their parents.

+ If r<=0.8, choose the first feasible insertion location. Else, choose the first entry. If infeasible, remove the offspring.

4.2.5. Mutation

We used swap mutation for the sequence chromosomes. In the swap mutation, two numbers are chosen at random, and their locations are switched. For our problem, this

method can be done in three steps: select individual, select random route, choose and swap two random customers.

4.2.6. Survivor selection

The best individual in a population is the one with the lowest fitness value. The fitness value in a genetic algorithm denotes the effectiveness or appropriateness of a particular solution to the issue at hand. The fittest individual in the population is chosen after each member of the population has had their fitness levels assessed. It stands for the evolutionary process's most ideal answer to date. The most successful individual is of tremendous importance since it offers insightful information about the caliber of solutions attained and acts as a model for future generations to improve upon.

4.3. Data Input

Assuming the units of information in our dataset:

- Time: minute
- Distance: kilometer
- Demand: unit
- Vehicle speed: 5/6 kilometer/minute

Our sample dataset was divided into three categories:

4.3.1. Customers

There is total 420 customers with seven types of information:

Cus No.: Customer number

X: x coordinate represents for the longitude

Y: y coordinate represents for the latitude

Service duration: the time served at each customer

Demand: the demand of each customer in units

Start: beginning of time window (earliest time for start of service), if any

End: end of time window (latest time for start of service), if any

Table 4.1: Example of Customers dataset

Cus No	X	Y	Service duration	Demand	Start	End
1	61.698	-53.092	25	25	274	599

2	-63.955	-22.283	3	2	197	420
3	75.778	-69.433	21	1	93	420
418	-77.422	71.717	25	1	71	380
419	71.045	71.232	16	2	97	368
420	-19.769	39.49	21	4	141	403

4.3.2. Depots

There are 12 depots with equal capacity including four types of information:

Depot: Depot number

X: x coordinate represents for the longitude

Y: y coordinate represents for the latitude

Capacity: maximum capacity of each depot

Table 4.2: Depots

Depot	X	Y	Capacity
1	44.306	-25.952	1000
2	-32.403	14.042	1000
3	37.663	13.769	1000
4	-15.104	12.111	1000
5	32.413	24.218	1000
6	10.846	-42.185	1000
8	-33.128	-14.172	1000
10	-25.037	19.618	1000
11	23.605	39.795	1000
12	41.196	45.258	1000

4.3.3. Vehicles

There is total 36 vehicles with three types of information:

Vehicle: Vehicle number

Max Load: maximum load of each vehicle

Max duration: maximum duration of each route

4.3.4. Assumption

The units of information in our dataset:

- Time: minute

- Distance: kilometer
- Demand: unit
- Vehicle speed: 2

Assume:

- Each day timeline starts at 8 am means that it will be 0 at initial point. For example, the 1st customer has a starting time and ending time at 274 and 599 respectively, it means that, this customer started at 12:34 and ended at 17:59.
- Vehicles are identical and each of them has 475 units of maximum load and 480 minutes of maximum duration.
- Since the dataset does not explicitly identify a beginning point, we can rely on a standard convention like the origin (0, 0). It might be regarded as the starting point. In this situation, the x-coordinate indicates horizontal displacement from the origin, while the y-coordinate represents vertical displacement.

4.3.5. Load data to Python

The load data process involves reading and processing instance data from a file. The data file includes information regarding the problem instance, including the number of vehicles, customers, and depots as well as the details of each customer, depot, and vehicle. The steps involved in loading data may be summarized up as follows:

- Open the data file. Read every line of the data file, then put the data in the proper data structures.
- The first line of the file should be used to extract the quantity of cars, clients, and depots.
- To store information on customers, depots, and vehicles, create empty lists.
- Store the maximum time and maximum load for each vehicle from the appropriate lines in the file.
- Store the properties for each client from the relevant lines in the file, including the index, coordinates, service time, demand, ready time, and due time.
- Store the information from the matching file lines for each depot's properties (index, location, and maximum capacity).

- Add the dictionaries for the information about the vehicle, the client, and the depot to the appropriate lists. Close the data file.

Following the load data procedure, we will have the following information:

- The quantity of cars, clients, and depots.
- Lists with dictionaries for each vehicle, client, and depot's characteristics.

Chapter 5. Result Analysis and Development

5.1. Initial Model Result

In the initial test model, we focus on the application of genetic algorithm and solution analysis using data given in Appendix I and parameter in section 5.3. However, there are three things need to notice. First, because our target is end-users who do not have any knowledge about algorithms, our analysis focuses on total distance and route plotting only. Second, we do not compare our algorithm with any in the papers due to our programming style is far different from the papers. And third, it is necessary to plot by coordinates because our data is obtained from the VRP-REP. By setting the parameters and based on default of the algorithm I obtained the following results:

- Depot 1: Three routes

Route 1: [421, 1, 266, 379, 85, 173, 190, 19, 421]

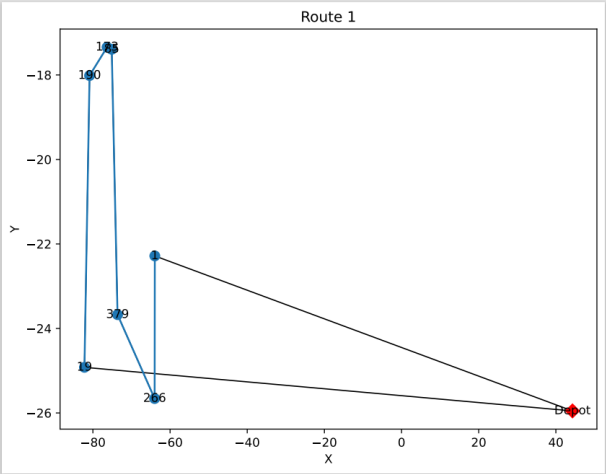

Figure 5.1: Depot 1 – Route 1

Route 2: [421, 102, 132, 396, 155, 229, 119, 355, 300, 212, 148, 150, 421]

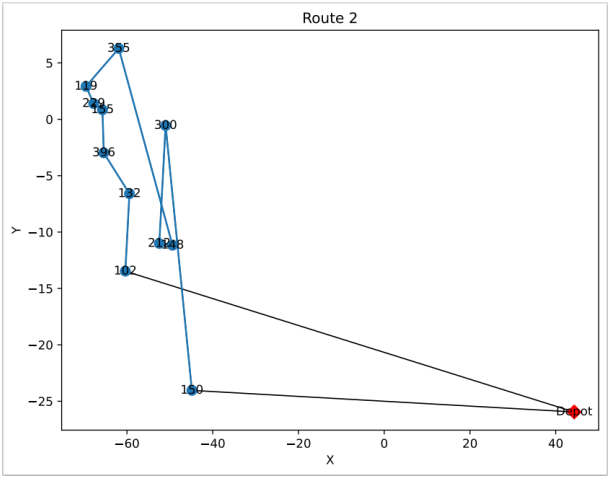

Figure 5.2: Depot 1 – Route 2

Route 3: [421, 305, 349, 243, 59, 26, 258, 421]

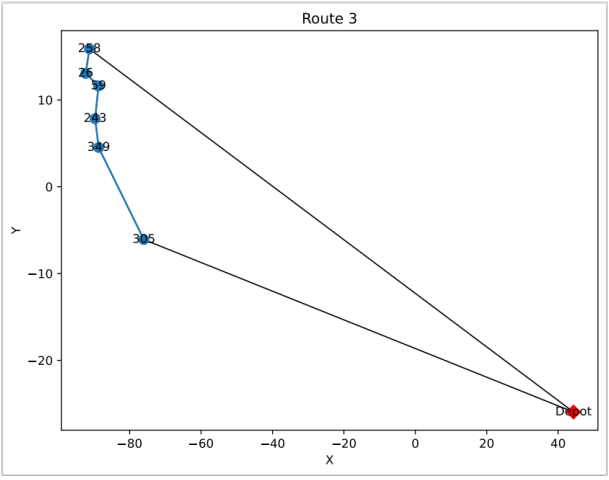

Figure 5.3: Depot 1 – Route 3

- Depot 2:

Route 1: [422, 2, 251, 105, 17, 339, 57, 32, 270, 422]

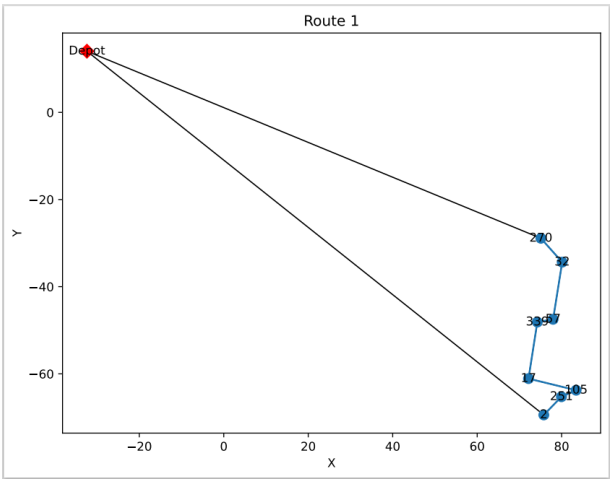

Figure 5.4: Depot 2 – Route 1

Route 2: [422, 66, 95, 217, 44, 309, 110, 192, 100, 402, 321, 422]

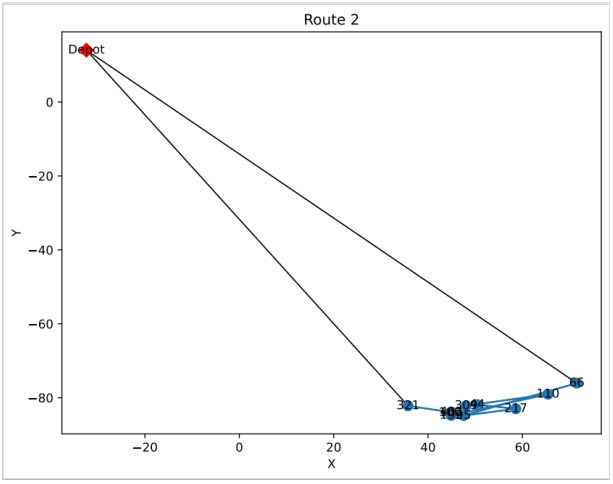

Figure 5.5: Depot 2 – Route 2

Route 3: [422, 372, 422]

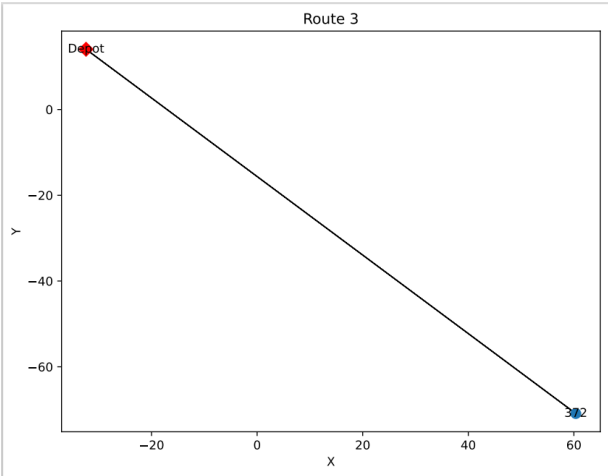

Figure 5.6: Depot 2 – Route 3

- Depot 3:

Route 1: [423, 3, 268, 42, 325, 38, 80, 104, 175, 315, 286, 128, 334, 423]

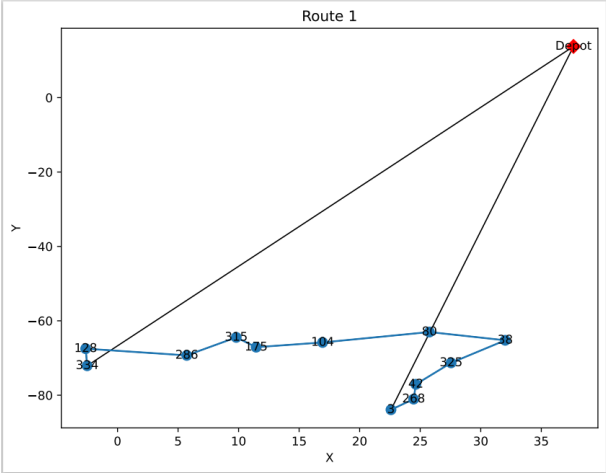

Figure 5.7: Depot 3 – Route 1

Route 2: [423, 256, 215, 328, 257, 284, 380, 375, 318, 144, 82, 423]

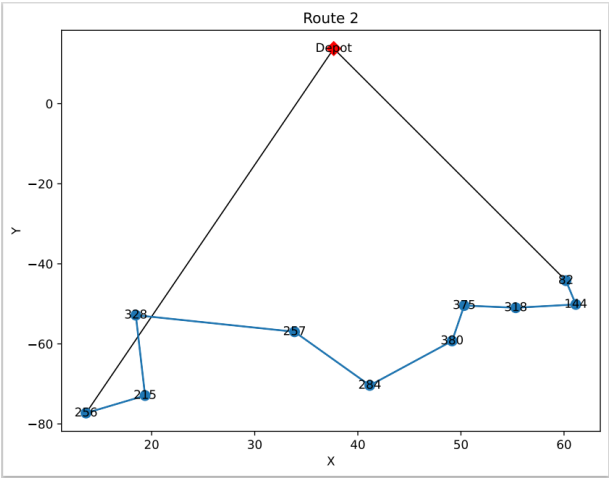

Figure 5.8: Depot 3 – Route 2

Route 3: [423, 153, 356, 388, 351, 423]

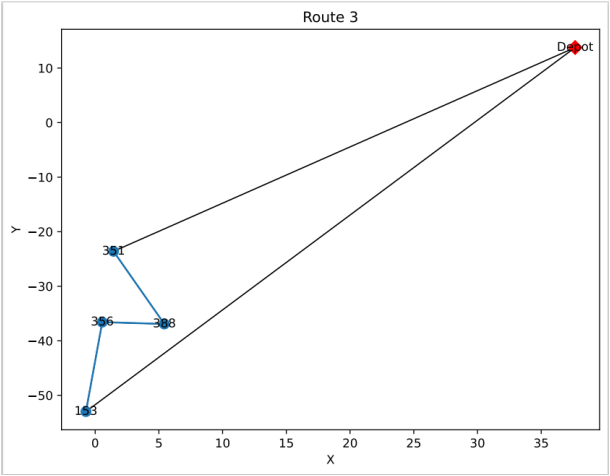

Figure 5.9: Depot 3 – Route 3

- Depot 4:

Route 1: [424, 4, 404, 178, 365, 301, 424]

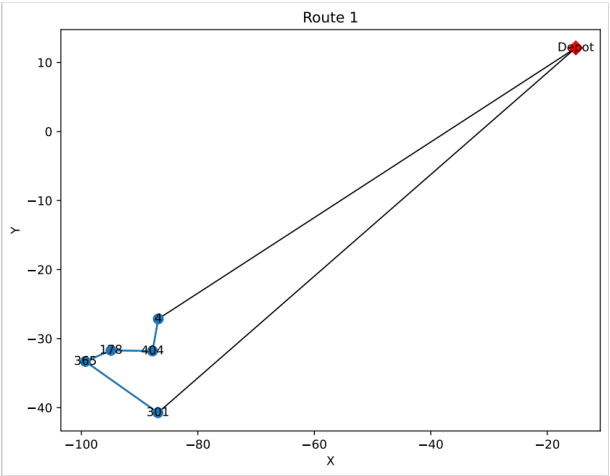

Figure 5.10: Depot 4 – Route 1

Route 2: [424, 277, 288, 167, 195, 385, 391, 255, 156, 117, 298, 424]

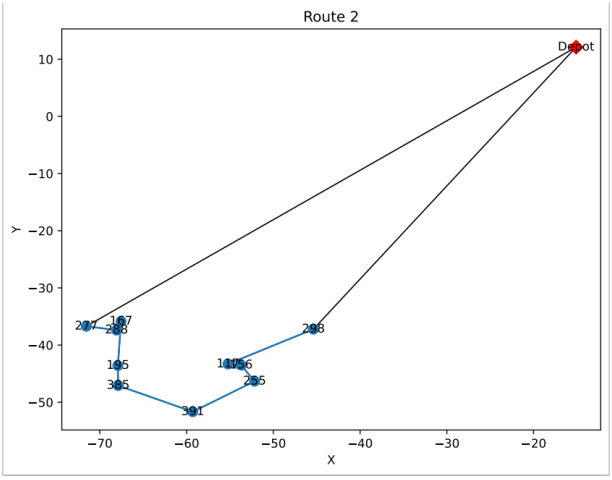

Figure 5.11: Depot 4 – Route 2

Route 3: [424, 70, 323, 424]

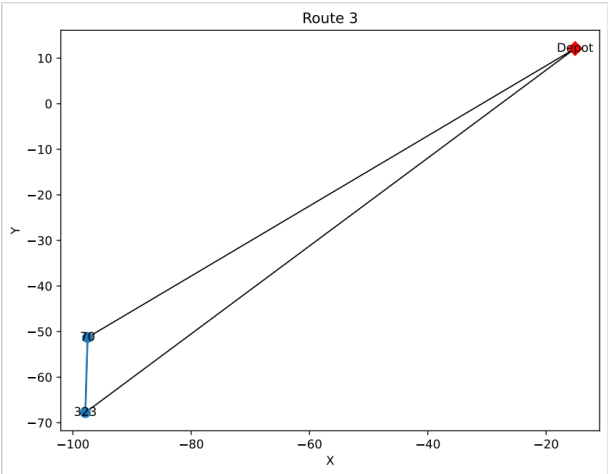

Figure 5.12: Depot 4 – Route 3

- Depot 5:

Route 1: [425, 5, 193, 126, 46, 425]

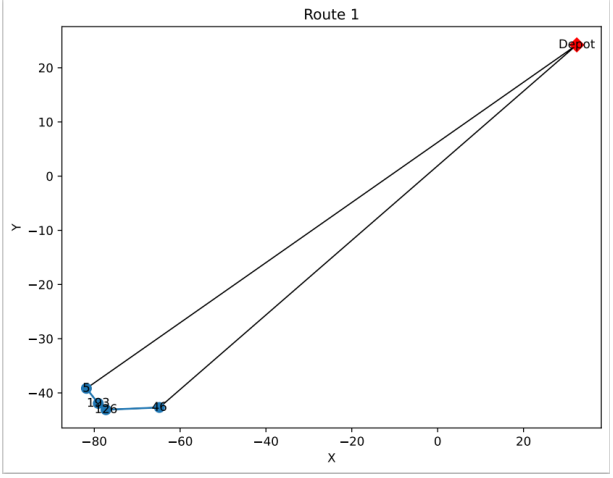

Figure 5.13: Depot 5 – Route 1

Route 2: [425, 15, 161, 364, 24, 67, 264, 425]

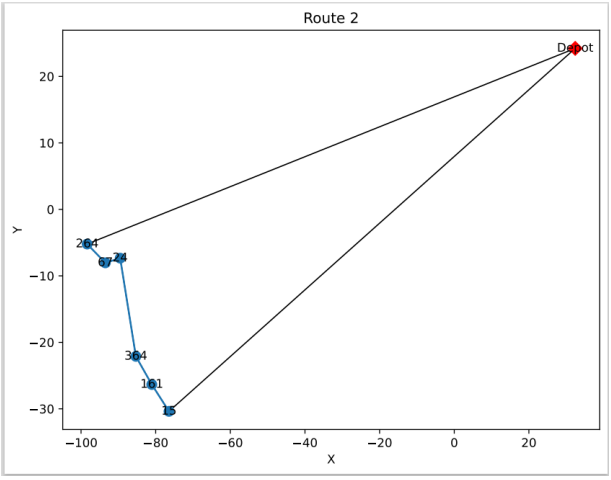

Figure 5.14: Depot 5 – Route 2

Route 3: [425, 345, 362, 160, 29, 280, 425]

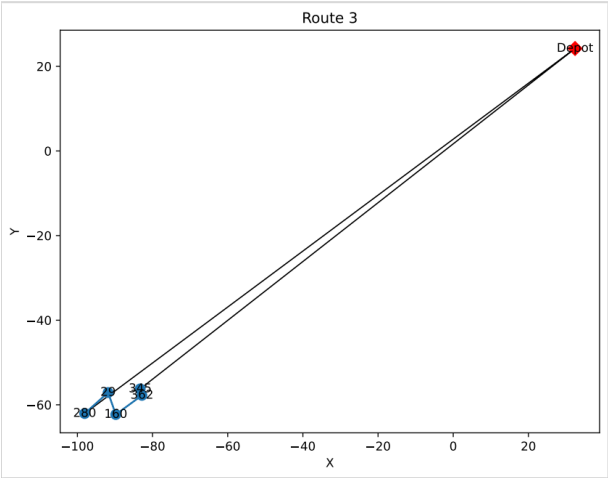

Figure 5.15: Depot 5 – Route 3

Route 4: [425, 184, 91, 245, 71, 308, 224, 152, 425]

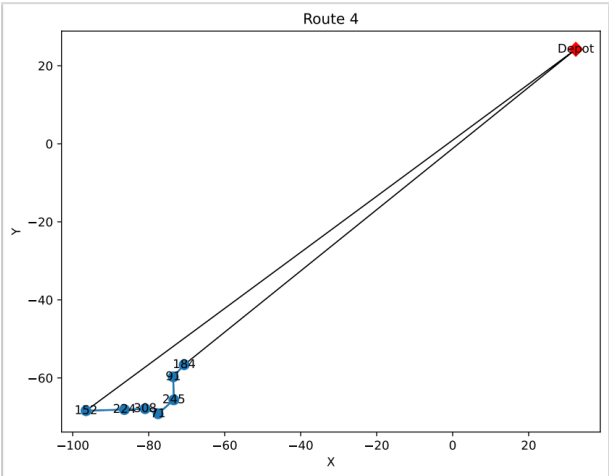

Figure 5.16: Depot 5 – Route 4

- Depot 6:

Route 1: [426, 6, 310, 387, 287, 225, 299, 317, 65, 327, 426]

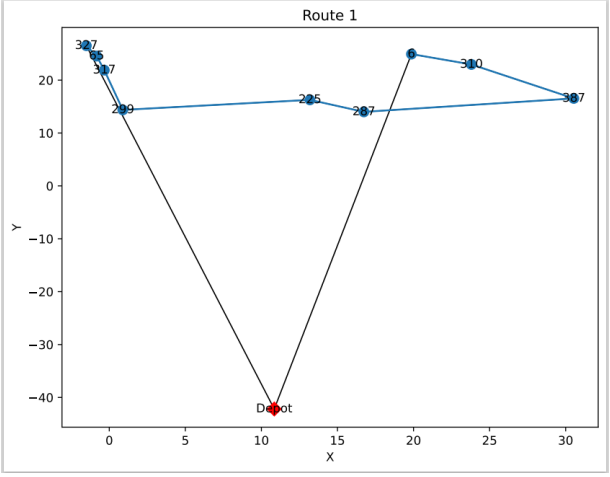

Figure 5.17: Depot 6 – Route 1

Route 2: [426, 403, 171, 183, 285, 426]

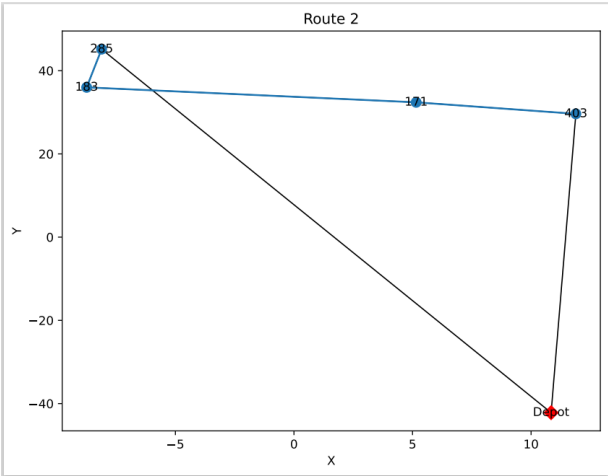

Figure 5.18: Depot 6 – Route 2

Route 3: [426, 63, 21, 279, 267, 259, 163, 426]

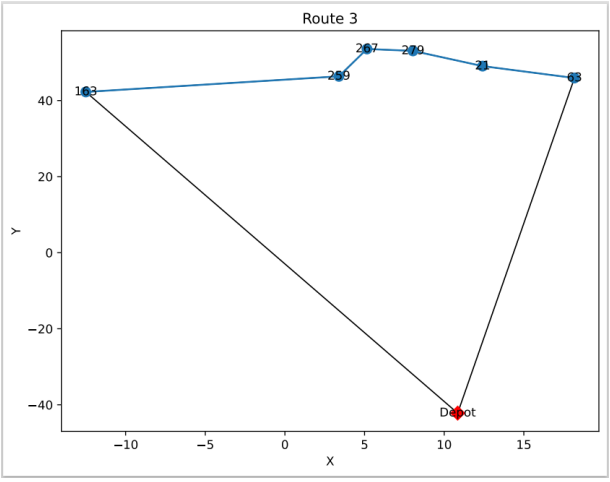

Figure 5.19: Depot 6 – Route 3

- Depot 7:

Route 1: [427, 7, 242, 314, 218, 427]

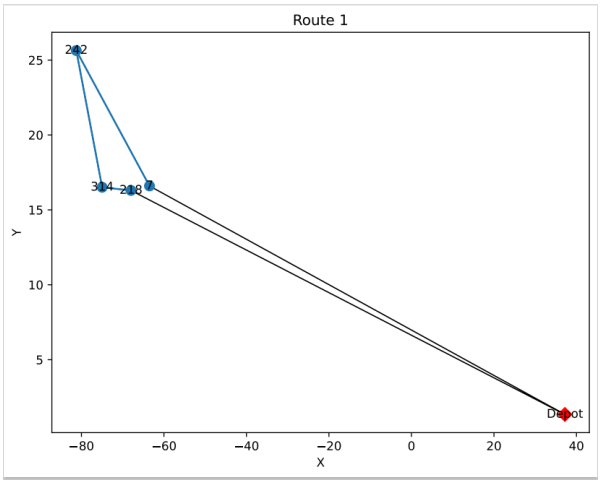

Figure 5.20: Depot 7 – Route 1

Route 2: [427, 326, 320, 187, 68, 357, 378, 39, 253, 236, 386, 56, 427]

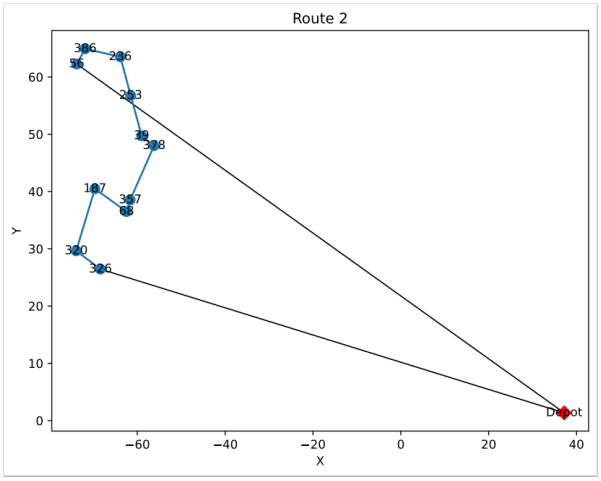

Figure 5.21: Depot 7 – Route 2

- Depot 8:

Route 1: [428, 8, 211, 55, 428]

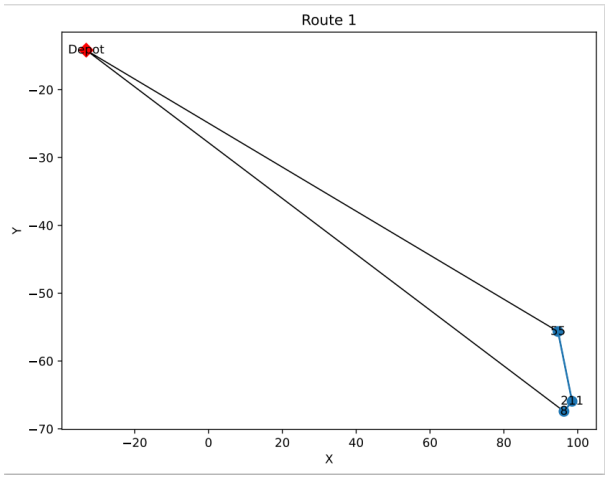

Figure 5.22: Depot 8 – Route 1

Route 2: [428, 337, 34, 428]

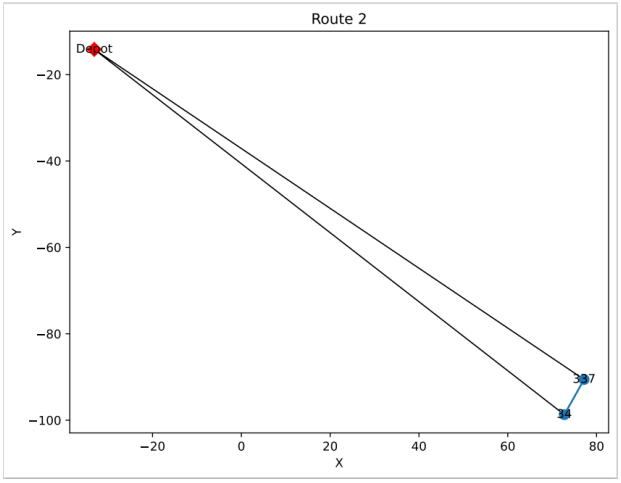

Figure 5.23: Depot 8 – Route 2

Route 3: [428, 392, 319, 341, 262, 176, 428]

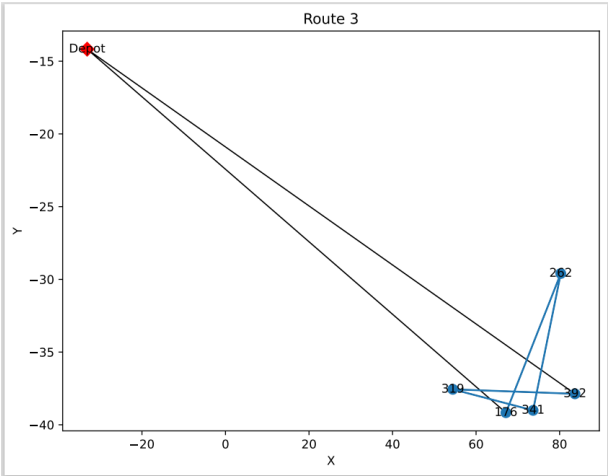

Figure 5.24: Depot 8 – Route 3

- Depot 9:

Route 1: [429, 9, 90, 129, 232, 429]

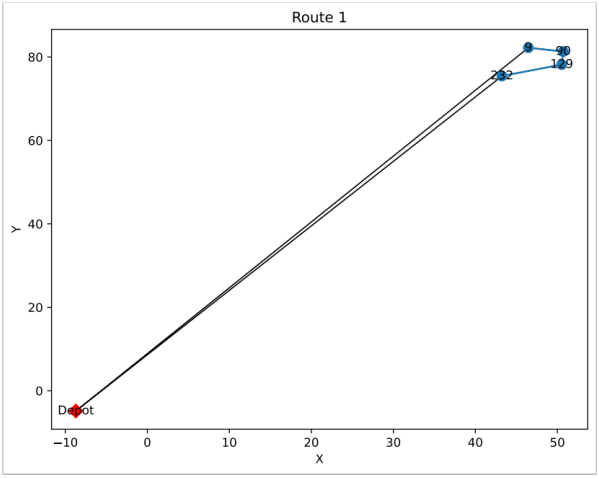

Figure 5.25: Depot 9 – Route 1

Route 2: [429, 114, 73, 99, 168, 133, 214, 36, 203, 429]

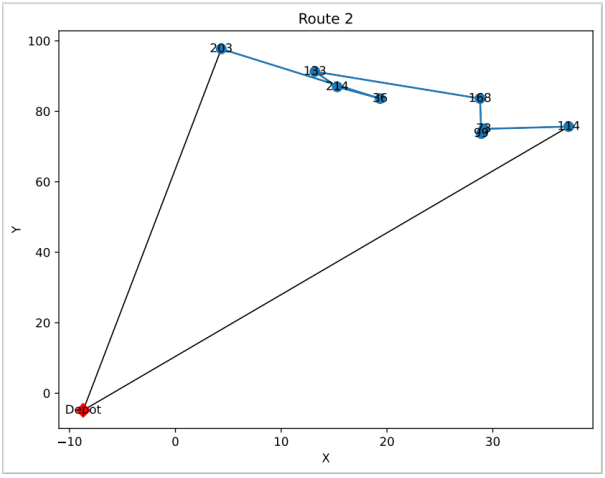

Figure 5.26: Depot 9 – Route 2

Route 3: [429, 381, 248, 13, 237, 186, 31, 81, 196, 429]

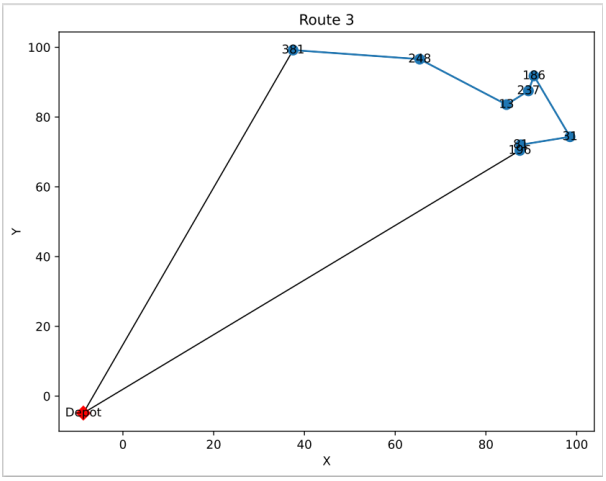

Figure 5.27: Depot 9 – Route 3

- Depot 10:

Route 1: [430, 10, 52, 54, 79, 430]

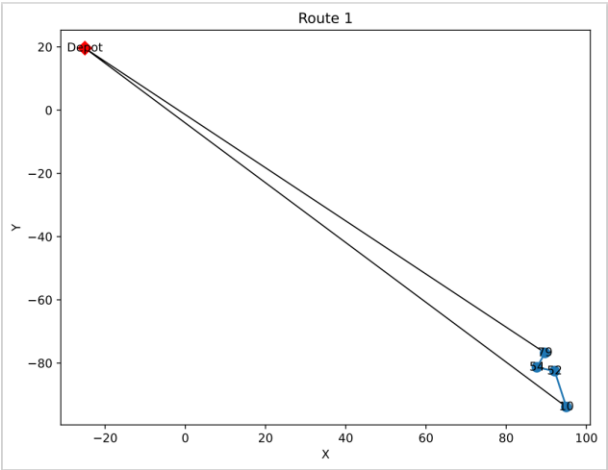

Figure 5.28: Depot 10 – Route 1

Route 2: [430, 239, 145, 399, 194, 188, 241, 430]

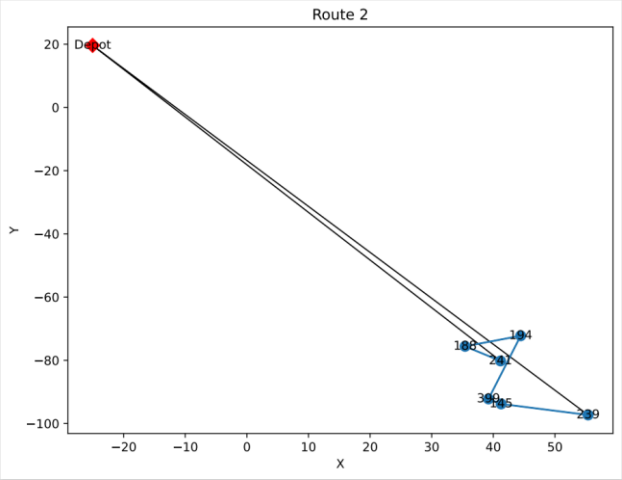

Figure 5.29: Depot 10 – Route 2

Route 3: [430, 0, 363, 234, 69, 377, 430]

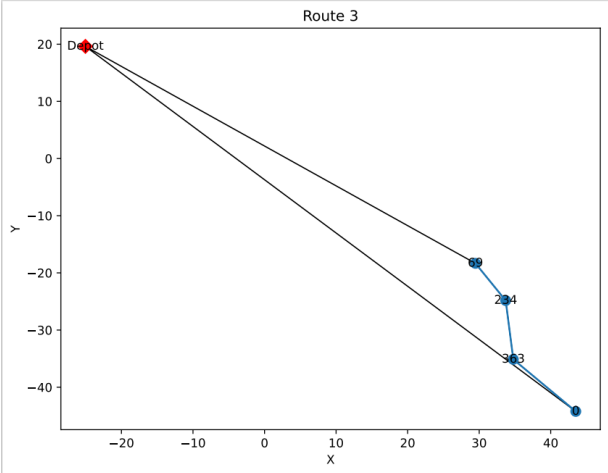

Figure 5.30: Depot 10 – Route 3

- Depot 11:

Route 1: [431, 11, 78, 112, 84, 382, 48, 140, 130, 431]

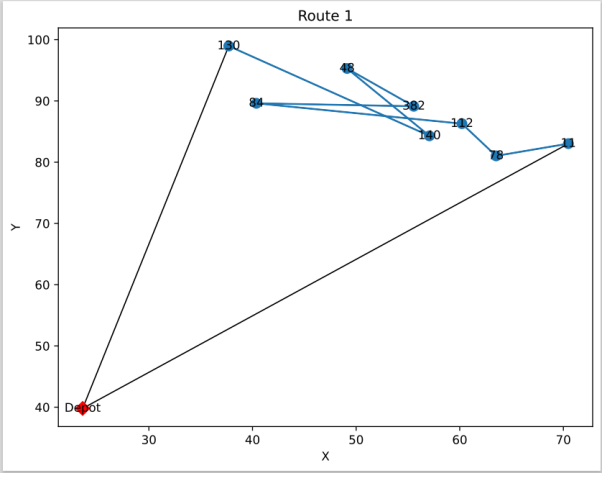

Figure 5.31: Depot 11 – Route 1

Route 2: [431, 86, 418, 376, 204, 28, 306, 416, 431]

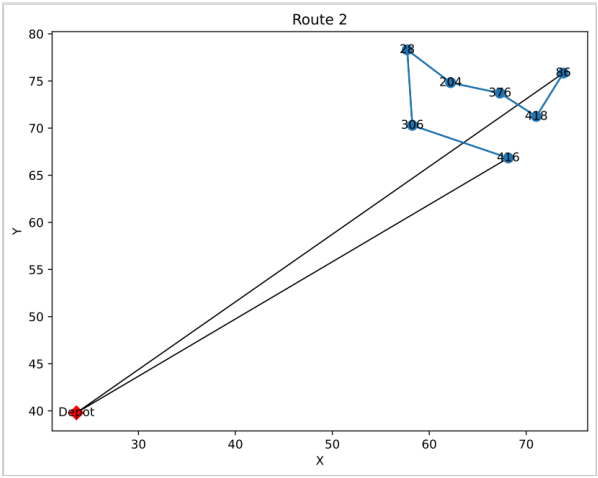

Figure 5.32: Depot 11 – Route 2

Route 3: [431, 374, 235, 290, 407, 207, 179, 276, 431]

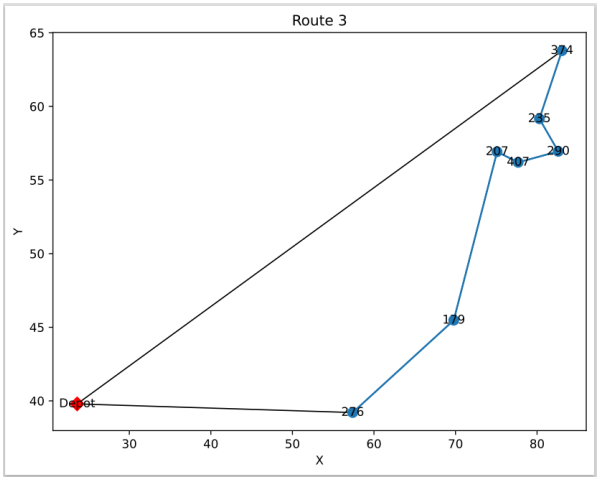

Figure 5.33: Depot 11 – Route 3

- Depot 12:

Route 1: [432, 12, 124, 113, 371, 108, 115, 432]

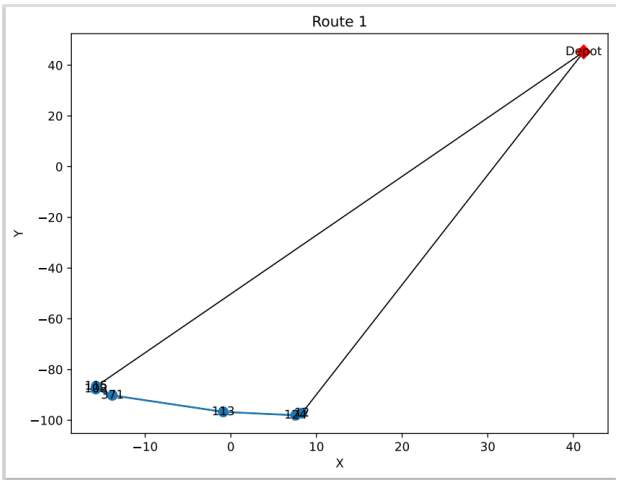

Figure 5.34: Depot 12 – Route 1

Route 2: [432, 94, 432]

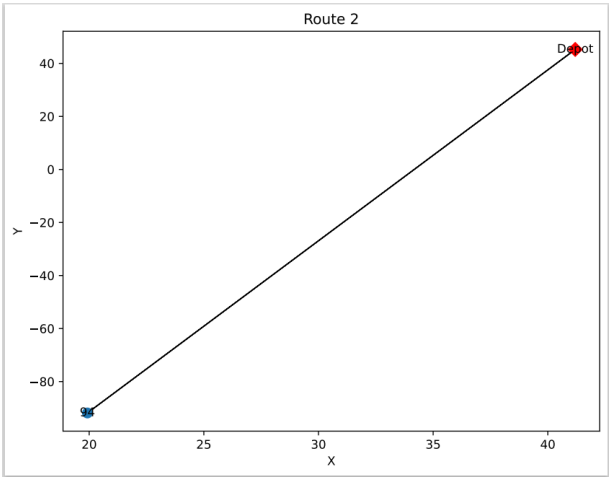

Figure 5.35: Depot 12 – Route 2

Route 3: [432, 201, 75, 185, 227, 281, 40, 394, 134, 30, 157, 246, 14, 405, 103, 400, 51, 432]

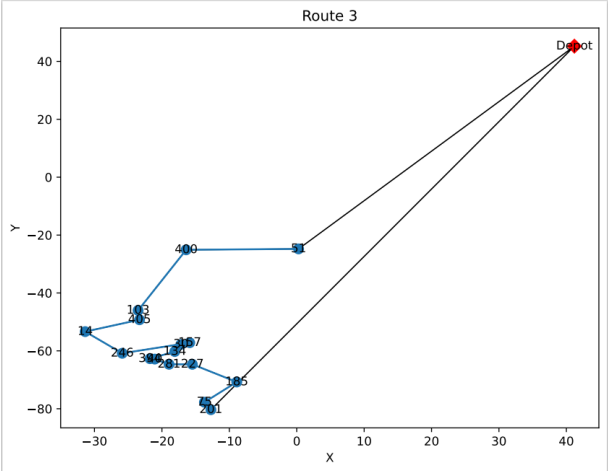

Figure 5.36: Depot 12 – Route 3

In summary, we have the table below that shows the number of customers and a list of total duration and total distance of each route.

Table 5.1: Summary the model results

		Number of customers	Total distance (km)	Total duration (minutes)
Depot 1	Route 1	7	267.3	477.54
	Route 2	8	265.354	440.545
	Route 3	6	294.608	463.371
Depot 2	Route 1	8	311.342	420.459
	Route 2	10	352.17	590.494
	Route 3	1	251.569	125.784
Depot 3	Route 1	12	259.118	486.641
	Route 2	10	254.42	603.486
	Route 3	4	164.298	126.608
Depot 4	Route 1	5	201.904	219.128
	Route 2	10	183.922	281.754
	Route 3	2	235.519	109.513
Depot 5	Route 1	4	267.191	423.305
	Route 2	6	292.977	307.141
	Route 3	5	320.777	237.307
	Route 4	7	324.535	433.855
Depot 6	Route 1	9	194.301	317.116
	Route 2	4	192.035	262.354
	Route 3	6	215.258	244.989

Depot 7	Route 1	4	246.067	209.329
	Route 2	11	303.857	478.456
Depot 8	Route 1	3	287.928	137.152
	Route 2	2	278.958	134.841
	Route 3	5	298.827	265.545
Depot 9	Route 1	4	214.01	238.414
	Route 2	8	263.464	359.053
	Route 3	8	329.769	428.283
Depot 10	Route 1	4	336.317	183.294
	Route 2	6	316.479	299.573
	Route 3	5	230.559	224.34
Depot 11	Route 1	8	220.153	401.257
	Route 2	7	153.214	424.504
	Route 3	7	140.654	178.612
Depot 12	Route 1	6	318.227	479.408
	Route 2	1	277.485	138.743
	Route 3	16	324.461	940.244
Total		229		

5.2. Loop Implementation

Because the output produced by GA is unstable, parallel programming is used to stabilize it. In contrast to parallel systems, a loop system has a loop whose limit is determined by users. The system runs the solver for each generation. The system first stores the top first-generation solution. The system then updates the current run best solution if the solution produced in the most recent generation is superior to that produced in the prior generation.

```
Best_solution_value = float (‘inf’)
For each run in max Number of generations:
        Execute GA
        If (GA.current_solution_value () < best_solution_value):
                best_solution_value = GA.getbest_solution_value ()
                best_solution = GA.getbest_solution_value()
```

Return best_solution

Other parameters of each generation:

- population_size = 50
- num_generations = 100
- tournament_size = 10
- crossover_rate = 0.8
- mutation_rate = 0.2
- threshold = 0.8
- penalty_weight_1 = 10000
- penalty_weight_2 = 10000
- penalty_weight_3 = 10000
- penalty_weight_4 = 10000
- penalty_weight_5 = 10000
- penalty_weight_6 = 10000
- penalty_weight_7 = 10000
- penalty_weight_8 = 10000

5.3. Convergence test of the Genetic Algorithm

We will analyze the convergence of our algorithm for the MDVRPTW through 100 generations. By plotting the best fitness value obtained in each generation, we can observe how the GA's performance improves over generations.

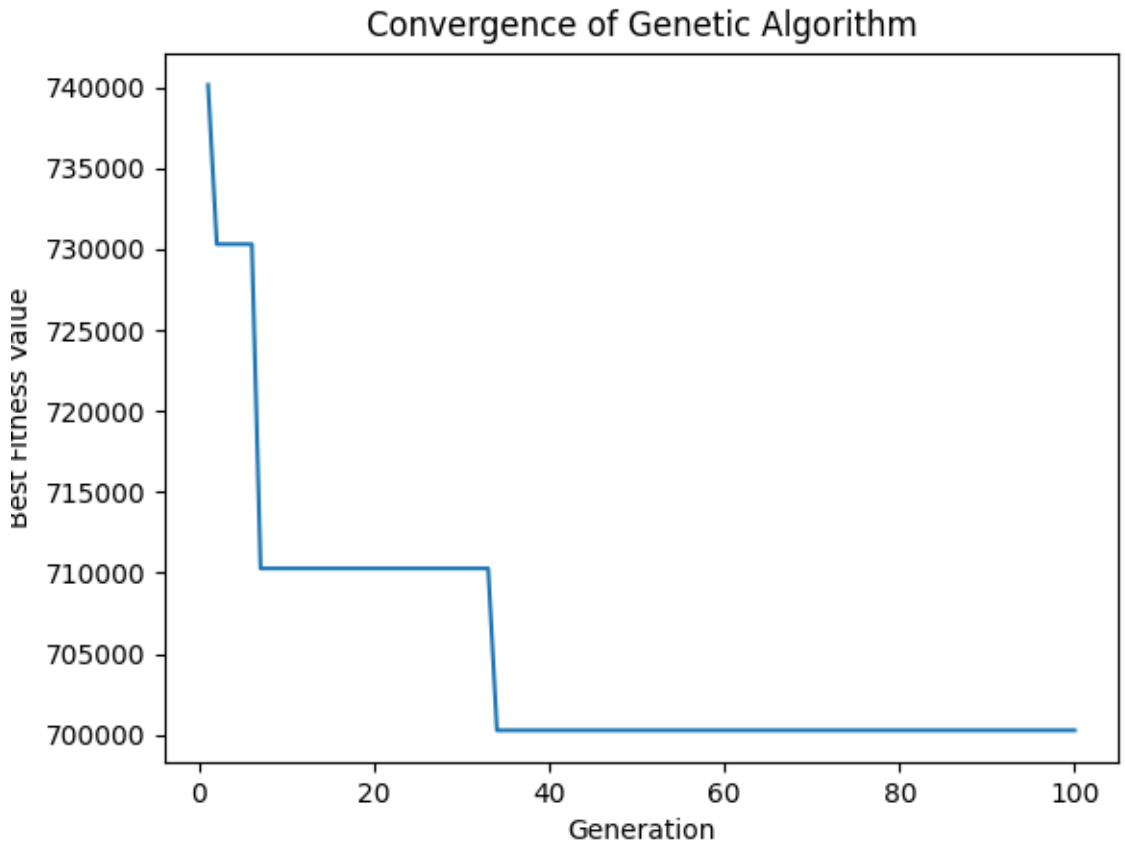

Figure 5.37: Convergence of GA

Statistical Value of Fitness value:

- Mean Fitness Value: 704874.9009782012
- Standard Deviation of Fitness Values: 8174.29433043459
- Minimum Fitness Value: 700274.7889194494
- Maximum Fitness Value: 740140.6542174496

We can observe the continuous improvement of the solution through each generation. And after generation 30, the fitness value gradually reaches the minimum value which means the solution is getting near optimal.

From this, we can conclude that the number of runs of GA will take into account the near optimal solution of the MDVRPTW.

5.4. Parameter tuning examination

We do changes with two important parameters of GA that is the population size and the

tournament size (with number of runs is 10)

- Population size:

Table 5.2: Effect of population size

Population size	Mean Fitness value	Standard deviation
25	710301.8278847651	0
50	700215.2580632765	9964.316273910044
75	692274.7889194494	13031.51457566449
100	670297.8170974598	21145.65266898466

Conclusion: While a lower population size could speed up convergence, it faces the risk of being caught in local optima due to its restricted exploration potential. A greater population size often enhances variety within the population.

- Tournament size:

Table 5.3: Effect of tournament size

Tournament size	Mean Fitness value	Standard deviation
5	724280.8244900916	4845.444722248148
10	700215.2580632765	9964.316273910044
15	702215.264247803	8386.971982510364
25	690301.8278847651	0

Conclusion: The convergence of solutions is accelerated by a higher tournament size. A smaller tournament can encourage variety, though.

5.5. Changing the number of vehicles

We change the number of vehicles to determine the relationship between the number of vehicles with the number of customers that we can serve.

We have the table below:

Table 5.4: Number of customers served change.

Number of vehicles	Number of customers served
27	168

36	235
45	288

Conclusion: The number of customers served increases when we increase the number of vehicles.

Chapter 6. Conclusions

6.1. Result Discussion

In this work, we tackled the Multi-Depot Vehicle Routing Problem with Time Windows (MDVRPTW), a complex and significant problem in logistics and transportation management. The major goal was to create a practical optimization strategy for successfully routing numerous cars among several depots while accommodating customers' time window restrictions. We used a Genetic Algorithm (GA) as a metaheuristic optimization tool to accomplish this.

The GA-based approach's results indicated its usefulness in dealing with the MDVRPTW. Taking into account customer demands, time windows, and vehicle capacity, the GA effectively developed viable and near optimum routes for each vehicle of each depot. The GA offered solutions that dramatically increased logistics efficiency and resource usage by reducing total travel distance and respecting customers' time frames.

In addition, we also test the performance of the algorithm through severe runs of generation, the result obtained is a good base for determining the efficiency and optimality of Genetic Algorithms.

Regarding the algorithm and coding method utilized for the solution, Genetic Algorithms may not be the most accurate and fastest method for optimization, but their adaptability to different types of problems is what makes them so popular and useful.

6.2. Limitation of the study and Recommendations for future research

- Problem Complexity: The MDVRPTW is a difficult optimization problem with many depots, vehicles, consumers, and time frame limitations. Finding globally optimum solutions for large-scale examples can therefore be difficult and time-consuming computationally. To enhance solution quality and shorten computation time, future researchers might investigate sophisticated optimization strategies like metaheuristics using hybrid approaches or machine learning algorithms.
- Parameter Tuning: The performance of the Genetic Algorithm (GA) utilized in this work is strongly dependent on the right tuning of parameters such as population

size, mutation rate, and crossover probability. Various parameter settings might provide different outcomes. Future researchers can carry out sensitivity studies and use cutting-edge methods, including adaptive parameter tuning, to improve the performance and resilience of the GA.

- Real-World Constraints: Although this research took into account basic restrictions like client requirements and time limits, real-world logistics situations may contain extra complications, such traffic conditions, vehicle capacity, and delivery priority. These real-world limitations can be added to the optimization model in future studies to make the suggested solutions more useful and applicable.
- Benchmarking and comparison: Although the GA-based technique for MDVRPTW showed promising results, it is important to compare and assess its performance against other cutting-edge algorithms and methods. Future researchers can compare various optimization approaches to evaluate their efficacy and efficiency and determine which ones are the most beneficial.

6.3. Timeline

	Task name	Start	End	Duration	Week																
					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Problems Identification																					
	Identify problem	0	2	2																	
	Literature Review	2	4	2																	
Design Concepts Consideration																					
	Advantages and disadvantages of considerd approaches	3	6	3																	
	Select the final methods	6	7	1																	
	Proposal Defense	7	8	1																	
Model Design																					
	Analyze initial model	8	9	1																	
	Model Development	9	12	3																	
	Feedback from advisor	10	12	2																	
Implementation																					
	Identify parameters and collect data	11	13	2																	
	Build the prototype for the process	11	14	3																	
Results Analysis																					
	Conduct validation	13	14	1																	
	Recheck again the results and methods	14	16	2																	
Complete and review the project																					
	Edit and finalize the report	14	17	3																	
Report Writing																					
	Writing report in academic form	3	16	13																	
Presentation																					
	Prepare the slides for presentation	16	17	1																	

Figure 6.1: Timeline of the thesis

6.4. Implications of the study

The use of Genetic Algorithms (GAs) to solve the Multi-Depot Vehicle Routing Problem with Time Windows (MDVRPTW) has important ramifications for a number of areas of the economy, society, and environment. By improving the effectiveness of logistics and transportation systems, GAs, a potent optimization tool, have the ability to reduce firms' overall operational costs by reducing fuel usage, vehicle maintenance, and other costs. From an economic perspective, the implementation of GA-based solutions in the MDVRPTW can result in streamlined routing and scheduling, reducing the number of vehicles required for delivery operations. As a result, businesses and customers could both benefit from decreased transportation costs, greater resource allocation, and increased delivery efficiency. Additionally, GAs' proficiency with complicated limits and time windows guarantees that deliveries are fulfilled within predetermined deadlines, minimizing fines and raising client satisfaction.

On a societal level, using GAs to solve the MDVRPTW results in better traffic control and less congestion in metropolitan areas. This can lessen commuter stress from traffic, lower greenhouse gas emissions from fewer idle vehicles, and create a more sustainable urban environment overall. Additionally, improved routing can lead to fewer collisions and safer roadways, improving general public safety.

From an environmental standpoint, logistics operations might potentially reduce their carbon footprint by employing GAs to optimize vehicle routes and timetables. Fuel consumption and emissions can be considerably decreased by cutting back on trip lengths and maximizing vehicle loads. This is consistent with the rising worldwide emphasis on resource conservation and environmental sustainability.

In conclusion, using genetic algorithms to solve the MDVRPTW offers significant advantages in the fields of economics, society, and the environment. This method improves efficiency, lowers costs, minimizes environmental impact, and supports safer and more sustainable transportation systems, making it a valuable solution in today's complex and interconnected world. It does this by optimizing vehicle routing, resource utilization, and delivery schedules.

REFERENCES

[1] Toth, P., & Vigo, D. (2002). 1. an overview of vehicle routing problems. *The Vehicle Routing Problem*, 1–26. https://doi.org/10.1137/1.9780898718515.ch1

[2] Czuba, P., & Pierzchala, D. (2021). Machine Learning methods for solving Vehicle Routing Problems. *Conference: 36th International Business Information Management Association (IBIMA)*

[3] Bae, H., & Moon, I. (2016). Multi-depot vehicle routing problem with time windows considering delivery and installation vehicles. *Applied Mathematical Modelling*, *40*(13-14), 6536–6549.

[4] Tan, K. C., Lee, L. H., Zhu, Q. L., & Ou, K. (2001). Heuristic methods for vehicle routing problem with time windows. *Artificial Intelligence in Engineering*, *15*(3), 281–295.

[5] Karimi, Behrooz & Javad, Mahsa. (2017). A simulated annealing algorithm for solving multi-depot location routing problem with backhaul. International Journal of Industrial and Systems Engineering. 25. 460. 10.1504/IJISE.2017.10003471.

[6] Karagul, K., Sahin, Y., Aydemir, E., & Oral, A. (2018). A simulated annealing algorithm-based solution method for a green vehicle routing problem with fuel consumption. *International Series in Operations Research & Management Science*, 161–187. https://doi.org/10.1007/978-3-319-97511-5_6.

[7] Wei, L., Zhang, Z., Zhang, D., & Leung, S. C. H. (2018). A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, *265*(3), 843–859. https://doi.org/10.1016/j.ejor.2017.08.035.

[8] Amine, K. (2019). Multi-objective Simulated Annealing: Principles and Algorithm Variants. *Advances in Operations Research*. 2019. 1-13. 10.1155/2019/8134674

[9] Kancharla, S. R., Ramadurai, G. (2020). Simulated annealing algorithm for multi depot two-echelon capacitated vehicle routing problem. *European Transport/Trasporti Europei*, *78*(ET.2020), 1–13. https://doi.org/10.48295/et.2020.78.8

[10] Hu, S.-L. (2010). Hybrid tabu search for the multi-depot vehicle routing problem. *SPIE Proceedings*. https://doi.org/10.1117/12.869032

[11] Zhang, L., & Niu, H. (2009). An algorithm for vehicle routing problem with soft time windows using tabu search. *ICCTP 2009*. https://doi.org/10.1061/41064(358)458

[12] Li, L., Chen, Y., & Meng, J. (2020). Improved tabu search algorithm for solving the vehicle routing problem with soft time windows in B2C environment. *2020 Chinese Control and Decision Conference (CCDC)*. https://doi.org/10.1109/ccdc49329.2020.9164444

[13] Ochelska-Mierzejewska, J., Poniszewska-Maranda, A., & Maranda, W. (2021). Selected Genetic Algorithms for Vehicle Routing Problem Solving. *Applied Sciences*, 11(5), 2179.

[14] Rybickova, A., Brodsky, J., Karaskova, A., & Mockova, D. (2015a). A genetic algorithm for the multi-depot vehicle routing problem. *Applied Mechanics and Materials*, *803*, 69–75. https://doi.org/10.4028/www.scientific.net/amm.803.69

[15] Ombuki-Berman, B., & Hanshar, F. T. (2009). Using genetic algorithms for multi-depot vehicle routing. *Bio-Inspired Algorithms for the Vehicle Routing Problem*, 77–99. https://doi.org/10.1007/978-3-540-85152-3_4

[16] [dataset] Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, *40*(1), 475–489. http://www.vrp-rep.org/datasets/item/2017-0013.html

APPENDIX I – DATA COLLECTED

Table 6.1: Customers

Cus No	X	Y	Service duration	Demand	Start	End
1	61.698	-53.092	25	25	274	599
2	-63.955	-22.283	3	2	197	420
3	75.778	-69.433	21	1	93	420
4	22.586	-83.907	14	14	251	545
5	-86.788	-27.145	18	5	154	504
6	-81.831	-39.166	14	4	216	412
7	19.863	24.925	16	7	96	371
8	-63.5	16.599	16	2	180	516
9	96.287	-67.422	17	15	157	452
10	46.465	82.227	22	13	235	492
11	94.992	-93.873	15	11	227	585
12	70.484	83.028	6	9	143	489
13	8.301	-97.197	17	11	173	509
14	84.535	83.563	16	11	236	561
15	-31.382	-53.353	10	25	70	377
16	-76.44	-30.377	24	13	276	507
17	98.978	24.314	16	2	145	369
18	72.157	-61.097	23	19	207	403
19	-52.145	-54.455	21	3	94	437
20	-82.192	-24.92	11	7	166	364
21	-34.223	-7.639	13	7	290	549
22	12.413	49.052	11	20	80	378
23	65.409	32.804	25	19	296	550
24	-49.305	-60.586	2	15	255	605
25	-89.514	-7.337	23	8	116	441
26	79.734	34.981	16	11	177	523
27	-92.319	13.061	11	6	298	519
28	-22	55.634	6	9	79	330
29	57.728	78.306	19	3	228	521
30	-91.824	-57.099	23	7	117	433
31	-17.173	-57.77	6	13	213	520
32	98.536	74.341	10	2	199	503
33	80.134	-34.37	8	6	162	439

34	-81.105	63.758	25	11	223	518
35	72.739	-98.718	24	18	224	491
36	-96.412	72.897	15	9	129	407
37	19.351	83.622	18	23	285	565
38	-95.833	32.825	20	15	276	477
39	32.024	-65.238	7	20	180	387
40	-59.011	49.719	23	14	207	517
41	-21.061	-62.843	19	3	80	349
42	-51.674	-88.581	8	8	206	490
43	24.641	-77.076	5	21	85	361
44	-83.013	88.023	20	15	143	381
45	50.435	-81.865	6	8	144	443
46	-49.351	11.697	2	15	277	555
47	-64.821	-42.687	3	11	226	437
48	78.986	5.467	10	7	230	494
49	49.127	95.298	19	18	265	505
50	12.855	-5.226	4	12	72	331
51	-18.273	68.258	9	3	254	509
52	0.272	-24.782	8	20	117	438
53	92.088	-82.543	4	7	231	476
54	92.321	-12.63	3	25	263	575
55	87.652	-81.329	22	7	291	547
56	94.671	-55.65	15	24	148	425
57	-73.793	62.268	3	4	115	334
58	77.979	-47.487	2	20	141	418
59	-50.435	-72.255	13	17	241	580
60	-88.727	11.618	4	3	100	382
61	-98.309	-98.872	4	1	242	441
62	-80.669	14.506	20	1	279	472
63	-40.123	-34.209	25	9	111	424
64	18.178	45.924	25	20	252	442
65	-33.767	-26.121	14	1	172	443
66	-0.83	24.55	14	1	195	435
67	71.51	-76.041	24	8	166	366
68	-93.487	-8.037	20	20	300	494
69	-62.462	36.497	18	14	197	470
70	33.705	-24.84	3	4	297	653

71	-97.518	-51.281	11	14	94	365
72	-77.523	-69.189	19	17	300	595
73	-66.109	-92.951	10	23	108	335
74	29.145	75.018	22	5	113	322
75	-43.613	98.877	1	22	112	378
76	-13.693	-77.74	12	14	189	499
77	-29.092	-73.952	4	15	101	284
78	-59.931	79.215	22	13	276	626
79	63.509	81.027	9	20	98	335
80	89.689	-76.777	21	21	200	544
81	25.765	-63.018	24	21	298	649
82	87.684	72.051	8	21	237	565
83	60.187	-44.204	12	5	212	522
84	-23.442	91.116	13	3	62	278
85	40.372	89.616	11	8	256	465
86	-75.122	-17.393	2	13	107	338
87	73.83	75.837	25	20	87	401
88	-74.374	86.281	20	7	157	355
89	-37.843	-84.149	21	13	116	423
90	19.354	-7.134	10	15	232	538
91	50.717	81.312	6	14	275	462
92	-73.54	-59.72	10	11	268	595
93	-28.217	-93.019	4	5	193	523
94	-19.089	57.813	24	23	178	530
95	19.923	-91.844	17	7	148	459
96	47.511	-84.832	23	24	111	381
97	-16.981	69.551	4	3	144	492
98	-18.664	33.174	13	9	70	422
99	-67.235	91.064	6	21	251	547
100	28.908	73.689	22	20	192	382
101	44.676	-84.053	1	16	283	510
102	8.477	-9.628	14	9	205	565
103	-60.378	-13.467	20	6	66	271
104	-23.535	-46.021	12	16	231	530
105	16.938	-65.821	10	21	240	573
106	83.435	-63.765	1	15	158	494
107	-60.326	-81.289	13	22	197	417

108	-51.889	-69.908	21	24	281	618
109	-15.767	-87.689	20	2	186	401
110	97.014	54.023	11	25	71	359
111	65.406	-79.043	13	5	200	469
112	-43.499	74.997	9	20	265	458
113	60.211	86.287	2	9	206	514
114	-0.885	-96.702	23	17	113	303
115	37.168	75.667	17	2	164	361
116	-15.723	-86.624	25	19	84	407
117	-11.834	23.295	16	23	229	477
118	-55.227	-43.313	18	13	211	543
119	-47.345	32.97	19	23	260	600
120	-69.543	2.916	1	6	184	414
121	-29.019	40.365	1	21	106	371
122	-2.71	93.995	6	2	233	524
123	-43.454	90.189	9	12	104	425
124	88.091	11.886	25	11	119	335
125	7.563	-98.034	19	3	96	335
126	-54.865	-79.097	17	19	280	573
127	-77.274	-43.105	3	4	253	527
128	-84.757	81.79	18	17	201	469
129	-2.629	-67.495	23	9	250	604
130	50.528	78.183	9	3	249	453
131	37.707	98.965	12	14	124	458
132	-95.286	62.158	12	18	171	479
133	-59.416	-6.597	17	16	96	334
134	13.186	91.277	24	16	215	434
135	-18.092	-60.229	13	23	297	622
136	75.747	12.289	2	18	293	481
137	-84.96	-86.749	15	21	155	407
138	-40.311	-76.732	6	3	185	371
139	97.655	61.4	24	6	102	355
140	-58.494	-74.726	9	18	105	339
141	57.07	84.299	21	9	252	542
142	-76.258	-98.432	5	16	150	390
143	-19.787	-16.937	20	2	180	462
144	-56.943	-91.321	11	16	251	448

145	61.143	-50.147	17	9	95	330
146	41.27	-93.797	15	8	171	466
147	0.765	68.327	16	12	79	386
148	22.652	-12.459	3	10	204	438
149	-49.373	-11.169	5	17	288	498
150	82.417	-12.685	10	24	172	388
151	-44.852	-24.038	18	17	189	414
152	-23.094	-82.944	11	22	175	387
153	-96.489	-68.407	16	15	171	352
154	-0.725	-52.985	4	17	159	507
155	77.578	22.849	7	15	243	556
156	-65.714	0.841	2	24	123	358
157	-53.707	-43.512	3	25	205	442
158	-15.875	-57.184	23	9	109	317
159	-51.839	-70.497	23	15	203	521
160	-98.22	42.876	25	16	202	412
161	-89.776	-62.296	8	12	127	427
162	-81.043	-26.338	15	20	226	519
163	-49.509	-87.155	17	14	159	370
164	-12.5	42.259	23	11	126	331
165	85.553	10.064	14	13	95	400
166	-16.781	-37.596	20	13	170	500
167	-39.146	93.141	9	10	178	476
168	-67.551	-35.844	13	16	86	332
169	28.793	83.633	14	6	191	491
170	-51.802	43.962	16	22	92	390
171	19.754	63.128	19	12	192	519
172	5.154	32.394	25	25	98	323
173	-67.541	90.141	13	7	82	271
174	-76.329	-17.343	7	1	199	423
175	88.549	28.026	5	6	152	499
176	11.451	-67.094	13	18	63	287
177	67.156	-39.168	10	22	71	266
178	74.817	-17.028	8	21	137	354
179	-94.906	-31.706	13	1	124	377
180	69.783	45.476	5	2	67	361
181	-99.324	90.437	21	25	241	535

182	2.646	-18.594	23	5	160	507
183	-98.794	97.084	17	4	115	393
184	-8.744	35.989	16	3	229	446
185	-70.657	-56.623	4	10	60	354
186	-8.899	-70.758	6	13	286	554
187	90.593	91.792	3	19	75	259
188	-69.661	40.475	10	8	98	410
189	35.422	-75.577	3	9	265	550
190	-13.872	52.023	21	4	233	440
191	-80.864	-18.014	14	23	119	429
192	-49.906	62.928	17	23	258	547
193	44.92	-84.784	10	11	83	321
194	-79.066	-41.935	25	16	245	546
195	44.448	-72.234	7	5	87	291
196	-67.918	-43.582	19	6	275	588
197	87.472	70.413	3	23	244	436
198	35.435	-5.409	8	5	105	441
199	78.102	-87.657	13	22	150	498
200	-69.842	-91.787	17	9	158	500
201	86.062	-87.463	4	12	210	408
202	-12.736	-80.323	5	19	257	488
203	-49.366	10.733	13	4	90	332
204	4.337	97.717	11	18	147	470
205	62.209	74.82	24	25	136	337
206	-37.401	95.287	7	13	195	507
207	-31.234	84.347	25	1	292	606
208	75.098	56.919	14	19	285	579
209	-40.335	18.837	4	16	255	541
210	77.002	18.75	6	1	268	603
211	-10.169	59.236	8	24	103	373
212	98.485	-65.939	18	20	270	603
213	-52.459	-11.012	20	18	142	418
214	-21.474	-95.484	11	18	173	386
215	15.295	86.941	20	14	221	491
216	19.363	-72.907	15	19	125	416
217	-30.405	63.795	14	3	113	342
218	58.593	-82.985	11	23	132	476

219	-67.986	16.297	25	15	263	596
220	-31.951	-79.813	12	10	247	574
221	-86.817	97.642	4	9	109	382
222	16.247	-10.061	9	16	233	592
223	-57.99	81.81	17	17	92	408
224	-78.809	40.467	16	1	107	296
225	-86.391	-68.107	8	12	93	434
226	13.192	16.269	16	6	146	381
227	-91.165	48.88	24	8	264	505
228	-15.488	-64.646	2	25	78	324
229	-18.252	65.455	7	15	253	516
230	-67.752	1.392	10	1	266	619
231	-85.39	64.069	14	1	240	497
232	-68.14	79.243	20	7	69	429
233	43.244	75.441	10	14	297	604
234	-53.04	73.969	20	4	154	356
235	34.769	-35.124	20	10	160	500
236	80.289	59.159	11	14	62	399
237	-63.871	63.521	15	12	162	437
238	89.372	87.509	20	19	264	488
239	-0.248	96.123	7	4	64	319
240	55.39	-97.293	18	15	103	427
241	-44.258	-60.83	19	7	155	422
242	41.174	-80.199	14	2	260	459
243	-81.241	25.64	11	11	199	505
244	-89.674	7.85	8	13	149	503
245	-91.676	46.272	3	5	153	372
246	-73.38	-65.62	15	23	261	454
247	-25.887	-60.834	8	18	187	407
248	-76.044	-90.242	4	24	142	487
249	65.403	96.59	2	6	102	369
250	-14.491	79.658	24	1	261	568
251	30.821	-11.012	16	3	239	458
252	79.954	-65.261	16	24	221	561
253	-80.909	99.697	1	2	198	475
254	-61.495	56.748	24	9	122	332
255	91.835	32.607	17	7	191	437

256	-52.159	-46.33	5	2	190	485
257	13.637	-77.285	17	6	278	559
258	33.86	-56.973	17	6	204	432
259	-91.274	15.887	5	20	73	391
260	3.389	46.365	25	6	221	480
261	-56.96	87.374	2	13	278	468
262	69.748	5.51	11	21	242	452
263	80.282	-29.594	25	13	228	554
264	22.879	-8.667	18	6	94	450
265	-98.36	-5.213	19	19	296	531
266	-92.319	17.497	22	9	92	409
267	-64.001	-25.659	12	3	300	480
268	5.152	53.556	16	15	280	486
269	24.457	-81.058	14	19	225	457
270	87.985	35.007	11	21	273	482
271	75.092	-28.896	6	3	296	523
272	-49.666	-43.556	19	9	66	332
273	-31.422	84.764	5	15	86	364
274	20.193	-14.287	8	12	293	563
275	-89.661	85.251	13	24	250	596
276	93.359	17.904	2	6	216	542
277	57.392	39.208	13	3	149	341
278	-71.557	-36.715	22	22	105	462
279	-14.524	89.167	12	24	181	504
280	8.039	53.054	1	6	110	461
281	-98.054	-62.054	3	7	184	486
282	-18.945	-64.643	25	15	268	608
283	-54.479	-87.044	17	1	220	545
284	-26.151	-94.12	9	5	83	374
285	41.159	-70.376	23	21	196	418
286	-8.113	45.164	2	4	87	325
287	5.719	-69.297	21	24	208	567
288	16.737	13.999	10	20	254	507
289	-68.07	-37.383	9	4	247	510
290	90.854	-12.755	21	22	242	508
291	82.599	56.945	9	24	231	413
292	-73.608	53.311	10	24	189	370

293	86.904	51.033	25	20	215	526
294	-38.314	-2.919	8	14	239	554
295	-31.951	90.2	14	17	211	393
296	-66.08	92.757	15	18	131	442
297	-61	-76.207	14	10	248	528
298	-70.4	51.207	8	19	178	377
299	-45.384	-37.226	4	6	89	321
300	0.891	14.371	11	12	158	410
301	-50.982	-0.558	2	12	110	374
302	-86.846	-40.739	22	24	253	552
303	-56.905	-90.4	16	25	80	342
304	-98.436	60.291	5	15	140	425
305	16.876	-25.749	19	6	262	591
306	-76.032	-6.08	21	6	64	361
307	58.256	70.29	21	7	187	405
308	-43.192	94.758	13	23	113	439
309	-80.935	-67.892	6	2	187	382
310	48.066	-82.159	7	4	108	403
311	23.805	22.964	25	10	110	366
312	-25.54	84.101	2	1	90	444
313	90.076	23.883	18	23	139	490
314	-49.712	76.162	15	2	102	382
315	-74.976	16.511	2	6	271	549
316	9.806	-64.46	4	3	290	643
317	-39.537	17.082	19	16	181	486
318	-0.326	21.885	14	1	141	400
319	55.316	-50.975	7	17	218	408
320	54.429	-37.561	23	14	180	489
321	-73.911	29.639	20	24	169	470
322	35.748	-82.187	6	3	168	504
323	58.624	36.732	21	24	202	527
324	-97.904	-67.77	25	17	227	407
325	82.049	11.889	25	6	232	585
326	27.552	-71.304	22	1	238	529
327	-68.397	26.445	24	23	237	518
328	-1.506	26.509	19	7	148	451
329	18.466	-52.826	15	12	290	549

330	-17.836	83.8	7	20	232	497
331	78.547	30.537	10	11	193	453
332	-81.505	79.09	4	2	224	531
333	-81.554	74.666	8	6	106	297
334	-47.398	-83.47	25	8	264	512
335	-2.507	-72.084	20	5	104	351
336	-16.36	99.113	1	25	269	603
337	-49.633	-42.145	22	14	73	279
338	77.193	-90.581	14	11	171	359
339	-37.834	-84.422	5	22	209	557
340	74.247	-48.119	24	24	74	296
341	16.36	-18.325	19	20	90	362
342	73.656	-39.008	22	23	201	475
343	-21.892	84.581	2	10	147	410
344	-11.789	-18.976	19	17	167	371
345	-16.962	43.168	25	1	155	342
346	-83.289	-56.239	9	13	153	430
347	67.037	4.975	17	2	82	277
348	-29.358	-85.877	21	12	207	502
349	-57.893	-88.917	6	10	224	562
350	-88.691	4.523	24	21	181	454
351	49.394	1.532	4	4	298	584
352	1.442	-23.599	21	23	222	559
353	19.846	-31.792	2	7	118	362
354	-81.837	67.918	19	20	100	327
355	57.841	10.655	20	18	296	648
356	-61.91	6.282	16	23	199	486
357	0.556	-36.577	14	7	267	531
358	-61.624	38.528	7	20	117	385
359	-38.309	-90.507	19	20	142	420
360	-98.35	-91.793	14	16	248	550
361	17.601	-11.336	13	9	271	628
362	87.995	8.526	13	19	255	501
363	-82.823	-57.76	25	9	107	453
364	43.5	-44.233	14	12	110	436
365	-85.31	-22.107	23	4	185	366
366	-99.301	-33.313	14	24	63	335

367	-3.536	52.261	23	20	137	331
368	-31.742	61.593	10	4	125	369
369	-69.25	84.881	16	12	62	293
370	2.1	-20.161	21	20	113	446
371	97.266	14.248	7	21	86	362
372	-13.831	-90.172	6	3	298	553
373	60.366	-70.903	13	20	230	545
374	51.401	-9.408	2	5	84	432
375	83.059	63.785	11	4	224	428
376	50.318	-50.419	14	22	158	431
377	67.298	73.711	19	4	183	423
378	29.476	-18.334	25	11	134	379
379	-56.177	48.002	14	10	245	471
380	-73.613	-23.673	5	14	274	614
381	49.143	-59.275	20	4	135	480
382	37.508	99.189	16	24	178	456
383	55.565	89.13	17	11	237	589
384	-37.53	52.127	23	18	191	381
385	-41.782	80.566	2	23	131	421
386	-67.887	-47.081	10	16	243	545
387	-71.875	64.92	11	13	116	397
388	30.537	16.541	13	21	105	435
389	5.429	-36.924	13	21	62	299
390	-19.466	-94.773	11	18	284	644
391	94.995	37.283	18	11	183	452
392	-59.284	-51.669	4	14	171	373
393	83.677	-37.87	25	23	255	585
394	-31.282	76.958	19	15	260	493
395	-21.775	-62.707	13	22	178	362
396	-38.303	79.185	1	18	99	379
397	-65.405	-2.976	20	9	294	502
398	-42.148	-93.52	14	6	64	342
399	-45.922	64.594	18	12	159	459
400	39.221	-92.221	13	24	183	479
401	-16.407	-25.092	1	20	158	379
402	-88.039	99.59	10	8	199	510
403	44.941	-83.928	18	14	200	558

404	11.889	29.626	15	1	216	558
405	-87.778	-31.8	23	11	189	420
406	-23.326	-49.252	4	10	282	487
407	44.533	15.419	16	15	134	356
408	77.683	56.199	18	2	183	398
409	-99.186	69.319	13	3	243	433
410	70.024	-10.589	12	9	91	290
411	-10.537	-8.642	21	6	284	640
412	5.848	4.905	14	6	294	636
413	-82.654	41.072	10	4	219	470
414	-27.619	-79.653	21	22	81	343
415	70.688	27.688	8	9	143	485
416	-37.749	-28.9	5	25	155	487
417	68.157	66.833	16	12	131	329
418	-77.422	71.717	25	1	71	380
419	71.045	71.232	16	2	97	368
420	-19.769	39.49	21	4	141	403

APPENDIX II – PYTHON CODE

```
import csv
import random
import math
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.path import Path
import matplotlib.patches as patches
#Function to calculate the travel time
def calculate_travel_time(start_node, end_node):
    # Calculate the travel time between two nodes (e.g., using
the assumed velocity)
    distance = calculate_distance(start_node, end_node)
    travel_time = distance / 2
    return travel_time

#Function to calculate the distance between two nodes
def calculate_distance(start_node, end_node):
    # Calculate the Euclidean distance between two nodes
    x1, y1 = start_node['x'], start_node['y']
    x2, y2 = end_node['x'], end_node['y']
    distance = ((x2 - x1)**2 + (y2 - y1)**2) ** 0.5
    return distance

# Step 1: Load data
# Function to load instance data from a file
def load_instance_data(instance_file):
    with open(instance_file, 'r') as file:
        lines = file.readlines()

    num_vehicles = int(lines[0].split()[1])
    num_customers = int(lines[0].split()[2])
    num_depots = int(lines[0].split()[3])

    customers = []
    depots = []
    vehicles = []

    # Extract vehicle data
    for i in range(1, num_vehicles+1):
        vehicle_data = lines[i].split()
        vehicle = {
            'Max Duration': int(vehicle_data[0]),
            'Max Load': int(vehicle_data[1])
        }
        vehicles.append(vehicle)
```

```
    # Extract customer data
    for i in range(num_vehicles+1, num_vehicles +
num_customers+1):
        customer_data = lines[i].split()
        customer = {
            'index': int(customer_data[0]),
            'x': float(customer_data[1]),
            'y': float(customer_data[2]),
            'service_time': int(customer_data[3]),
            'demand': int(customer_data[4]),
            'ready_time': int(customer_data[-2]),
            'due_time': int(customer_data[-1]),
        }
        customers.append(customer)

    # Extract depot data
    for i in range(num_customers+num_vehicles+1,
num_vehicles+num_customers+num_depots+1):
        depot_data = lines[i].split()
        depot = {
            'index': int(depot_data[0]),
            'x': float(depot_data[1]),
            'y': float(depot_data[2]),
            'Max Capacity': int(depot_data[-1])
        }
        depots.append(depot)

    return num_vehicles, num_customers, num_depots, customers,
depots, vehicles

# Generate the distance matrix and travel time matrix
def generate_distance_matrix():
    num_nodes = num_customers + num_depots
    distance_matrix = [[0] * num_nodes for _ in range(num_nodes)]
    travel_time_matrix = [[0] * num_nodes for _ in
range(num_nodes)]
    # Calculate distances between customers
    for i in range(num_customers):
        for j in range(i+1, num_customers):
            distance = calculate_distance(customers[i],
customers[j])
            distance_matrix[i][j] = distance
            distance_matrix[j][i] = distance
            travel_time = distance / (2)  # Assuming velocity is
5/6
            travel_time_matrix[i][j] = travel_time
            travel_time_matrix[j][i] = travel_time
```

```
    # Calculate distances and travel time between depots and
customers
    for i in range(num_customers):
        for j in range(num_customers, num_nodes):
            distance = calculate_distance(customers[i], depots[j-
num_customers])
            distance_matrix[i][j] = distance
            distance_matrix[j][i] = distance
            travel_time = distance / (2)  # Assuming velocity is
5/6
            travel_time_matrix[i][j] = travel_time
            travel_time_matrix[j][i] = travel_time
    return distance_matrix, travel_time_matrix

def save_distance_matrix(distance_matrix, filename):
    with open(filename, 'w', newline='') as file:
        writer = csv.writer(file)
        for row in distance_matrix:
            writer.writerow(row)

def save_travel_time_matrix(travel_time_matrix, filename):
    with open(filename, 'w', newline='') as file:
        writer = csv.writer(file)
        for row in travel_time_matrix:
            writer.writerow(row)

# Step 2: Define parameters
population_size = 50
num_generations = 100
tournament_size = 10
crossover_rate = 0.8
mutation_rate = 0.2
threshold = 0.8
penalty_weight_1 = 10000
penalty_weight_2 = 10000
penalty_weight_3 = 10000
penalty_weight_4 = 10000
penalty_weight_5 = 10000
penalty_weight_6 = 10000
penalty_weight_7 = 10000
penalty_weight_8 = 10000

# Step 4: Define fitness calculation
def calculate_fitness(individual):
    total_distance = 0.0
    total_duration = 0.0
    visited_customers = set()
```

```
    penalty = 0.0
    for vehicle_route in individual:
        if len(vehicle_route) <=2:
            break
        else:
            total_distance, total_duration =
calculate_route_distance_duration(vehicle_route)

            # Constraint 1: Ensuring the vehicle that leaves the
customer is the same as the one that visits the customer
            if vehicle_route[0] != vehicle_route[-1]:
                penalty += penalty_weight_1

            # Constraint 2: Each customer is assigned to a
vehicle
            for i in range(1, len(vehicle_route) - 2):
                if vehicle_route[i] == vehicle_route[i + 1]:
                    penalty += penalty_weight_2

            # Constraint 3: Determining whether each customer is
assigned to a depot or not
            for customer in vehicle_route[1:-1]:
                if customer in visited_customers:
                    penalty += penalty_weight_3
                else:
                    visited_customers.add(customer)

            # Constraint 4: Maximum load for each vehicle
            total_demand = 0
            for i in range(len(vehicle_route) - 2):
                customer = customers[vehicle_route[i + 1]]
                total_demand += customer['demand']
                if total_demand > vehicles[0]['Max Load']:
                    penalty += penalty_weight_4

            # Constraint 5: Maximum duration for each vehicle
            if total_duration > vehicles[0]['Max Duration']:
                penalty += penalty_weight_5

            # Constraint 6,7,8: Time constraints between
consecutive customers
            current_time = 0.0
            for i in range(len(vehicle_route) - 1):
                if vehicle_route[i] > num_customers:
                    start_node = vehicle_route[i]
                    end_node = vehicle_route[i + 1]
                    current_time += travel_time_matrix[start_node
```

```
- 1][end_node]
                elif vehicle_route[i + 1] > num_customers:
                    start_node = vehicle_route[i]
                    end_node = vehicle_route[i + 1]
                    current_time +=
travel_time_matrix[start_node][end_node - 1]
                else:
                    start_node = customers[vehicle_route[i]]
                    end_node = customers[vehicle_route[i + 1]]
                    distance = calculate_distance(start_node,
end_node)
                    current_time += start_node['service_time']
                    current_time += distance / (2)
                    current_time = max(current_time,
end_node['ready_time'])
                    if 'ready_time' in end_node:
                        if current_time < end_node['ready_time']:
                            penalty += penalty_weight_6
                    if 'ready_time' and 'service_time' in
end_node:
                        if current_time +
end_node['service_time'] < end_node['ready_time']:
                            penalty += penalty_weight_7
                    if 'due_time' and 'service_time' in end_node:
                        if current_time +
end_node['service_time'] > end_node['due_time']:
                            penalty += penalty_weight_8
        fitness = total_duration + penalty
    return fitness

def calculate_route_distance_duration(vehicle_route):
    distance = 0
    duration = 0.0
    current_time = 0.0
    if len(vehicle_route) <= 2:
        distance = 0
        duration = 0
    else:
        for i in range(len(vehicle_route) - 1):
            if vehicle_route[i] > num_customers:
                start_node = vehicle_route[i]
                end_node = vehicle_route[i + 1]
                distance += distance_matrix[start_node-
1][end_node]
                duration += travel_time_matrix[start_node-
1][end_node]
            elif vehicle_route[i+1] > num_customers:
```

```
                start_node = vehicle_route[i]
                end_node = vehicle_route[i + 1]
                distance += distance_matrix[start_node][end_node-
1]
                duration +=
travel_time_matrix[start_node][end_node-1]
            else:
                customer1 = customers[vehicle_route[i]]
                customer2 = customers[vehicle_route[i + 1]]
                distance += calculate_distance(customer1,
customer2)
                current_time += distance/(4)
                current_time = max(current_time,
customer2['ready_time'])
                current_time += customer2['service_time']
                duration = max(duration, current_time -
customer2['ready_time'])
    return distance, duration
# Step 5: Generate initial population
def generate_initial_population(population_size, num_vehicles,
num_customers, customers, depots, vehicles):
    population = []
    for _ in range(population_size):
        remaining_customers = list(range(num_customers))
        random.shuffle(remaining_customers)
        random.shuffle(depots)
        individual = create_vehicle_sequence(num_vehicles,
remaining_customers, customers, depots, vehicles)
        population.append(individual)
    return population

def create_vehicle_sequence(num_vehicles, remaining_customers,
customers, depots, vehicles):
    vehicle_sequence = []
    depot_capacity = 1000
    for vehicle_idx in range(num_vehicles):
        depot_idx = vehicle_idx % len(depots)
        depot_index = depots[depot_idx]['index']
        sequence = [depot_index]
        maxLoad = vehicles[vehicle_idx]['Max Load']
        maxDuration = vehicles[vehicle_idx]['Max Duration']
        current_capacity = 0
        current_time = 0
        while remaining_customers:
            feasible_customers = []
            distances = []
            durations = []
```

```
            for customer in remaining_customers:
                if customer <= num_customers:
                    distance =
calculate_distance(customers[sequence[-1] - num_customers],
customers[customer])
                else:
                    distance =
calculate_distance(depots[depot_idx], customers[customer])
                arrival_time = current_time + distance / 4
                waiting_time = max(0,
customers[customer]['ready_time'] - arrival_time)
                service_time =
customers[customer]['service_time']
                update_current_time = arrival_time + waiting_time
+ service_time
                if (update_current_time <= maxDuration and
                        current_capacity +
customers[customer]['demand'] <= maxLoad and
                        current_capacity +
customers[customer]['demand'] <= depot_capacity):
                    feasible_customers.append(customer)
                    distances.append(distance)
                    durations.append(update_current_time)
            if feasible_customers:
                best_customer =
feasible_customers[np.argmin(distances)]
                sequence.append(best_customer)
                remaining_customers.remove(best_customer)
                current_capacity +=
customers[best_customer]['demand']
                current_time = durations[np.argmin(distances)]
            else:
                break
        sequence.append(depot_index)
        vehicle_sequence.append(sequence)
    return vehicle_sequence

# Step 6: Genetic operators (Selection, Crossover, Mutation)
def tournament_selection(population, fitness_values,
tournament_size,threshold):
    #choose random k individuals in the population
    k = tournament_size
    p = threshold
    new_gen = []
    for _ in range(k):
        tournament_indices =
random.sample(range(len(population)),k)
```

```
        tournament_fitness = [fitness_values[i] for i in
tournament_indices]
        sorted_indices = sorted(range(len(tournament_fitness)),
key=lambda x: tournament_fitness[x], reverse=True)
        probabilities = np.array([p * (1 - p) ** i for i in
range(k)])  # Compute the selection probabilities
        probabilities /= np.sum(probabilities)  # Normalize the
probabilities to sum to 1
        # Select the best individuals according to the
probabilities
        selected_index = np.random.choice(sorted_indices,
p=probabilities)

new_gen.append(population[tournament_indices[selected_index]])
    return new_gen

def best_cost_route_crossover(parent1, parent2):
    # Randomly select a depot
    depot = random.choice(depots)['index']
    # Randomly select p1,p2 from each parent
    p1 = random.choice(parent1)
    p2 = random.choice(parent2)
    #Randomly select a route from each parent, r1 from p1, r2
from p2
    r1 = random.choice(p1)
    r2 = random.choice(p2)
    # Remove all customers of route1 from parent2, and route2
from parent1
    new_p1 = remove_customer(p1,r2)
    new_p2 = remove_customer(p2,r1)
    #Recreate individual
    offspring1 = recreate_vehicle_sequence(new_p1, r2, depot)
    offspring2 = recreate_vehicle_sequence(new_p2, r1, depot)

    return offspring1, offspring2

def remove_customer(p, r):
    new_routes = []
    removed_customers = set()
    for route in p:
        new_route = route[:]
        for customer in r[1:-1]:
            if customer in new_route[1:-1]:
                new_route.remove(customer)
                removed_customers.add(customer)
        new_routes.append(new_route)
    for sequence in new_routes:
```

```
        if len(sequence) == 2:
            new_routes.remove(sequence)
    return new_routes

def recreate_vehicle_sequence(newp, remove_route, depot):
    num_routes = len(newp)
    if num_routes < num_vehicles:
        # Add new routes with the removed customers
        remaining_depots = [depot]
        remaining_customers = list(range(num_customers))
        random.shuffle(remaining_customers)
        for route in newp:
            remaining_depots.append(route[0])
        depot_info = []
        for i in remaining_depots:
            depot_info.append(depots[i-num_customers-1])
        # Adjust existing routes by inserting the removed
customers
        new_sequence =
create_vehicle_sequence(num_vehicles,remaining_customers,customer
s,depot_info,vehicles)
    else:
        remaining_depots = []
        remaining_customers = list(range(num_customers))
        random.shuffle(remaining_customers)
        for route in newp:
            remaining_depots.append(route[0])
        depot_info = []
        for i in remaining_depots:
            depot_info.append(depots[i-num_customers-1])
        # Adjust existing routes by inserting the removed
customers
        new_sequence =
create_vehicle_sequence(num_routes,remaining_customers,customers,
depot_info,vehicles)

    return new_sequence

def swap_mutation(individual):
    route = random.choice(individual)
    if len(route) >= 4:
        customer1, customer2 = random.sample(route[1:-1], 2)
        index1 = route.index(customer1)
        index2 = route.index(customer2)
        route[index1], route[index2] = route[index2],
route[index1]
```

```
    return individual

def bestIndividual(offspring):
    fitness_values = [calculate_fitness(individual) for
individual in offspring]
    best_index = min(range(len(offspring)), key=lambda i:
fitness_values[i])
    best_individual = offspring[best_index]
    return best_individual

def _get_edges_from_route(route):
    assert len(route) >= 2
    edges = []
    for i, n in enumerate(route):
        if i >= 1:
            edges.append((route[i - 1], route[i]))
    return edges

def draw_route(pos, route, names, route_number, save_file=False,
file_name=None):
    fig, ax = plt.subplots(figsize=(8, 6))

    # Set plot properties
    customer_indices = [n + 1 for n in route[1:-1] if n in pos]
# Exclude the depot index
    ax.scatter([pos[n][0] for n in customer_indices], [pos[n][1]
for n in customer_indices])
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_title(f'Route {route_number}')
    verts = [pos[route[0]]] + [pos[n] for n in customer_indices]
+ [pos[route[-1]]]  # Include the depot as the start and end
    path = Path(verts)
    patch = patches.PathPatch(path, facecolor='none', lw=1,
zorder=0)
    ax.add_patch(patch)
    # Get x and y coordinates for customers in the route (exclude
depot)
    x = [pos[n][0] for n in customer_indices]
    y = [pos[n][1] for n in customer_indices]
    ax.plot(x, y, marker='o', linestyle='-', markersize=8)

    # Add customer index labels to each customer location
    for i, (xi, yi) in enumerate(zip(x, y)):
        ax.text(xi, yi, str(route[i+1]), ha='center',
va='center', fontsize=10, color='black')  # Add 1 to index
```

```
    # Add the depot as the first and last point in the route
    depot_x, depot_y = pos[route[0]]
    ax.plot(depot_x, depot_y, marker='D', markersize=8,
color='red', label='Depot')
    ax.text(depot_x, depot_y, 'Depot', ha='center', va='center',
fontsize=10, color='black')

    # Show the plot for this route
    if save_file:
        if file_name is None:
            file_name = f"Route_{route_number}"
        plt.savefig(f"{file_name}.pdf", bbox_inches='tight',
transparent=True, pad_inches=0.1)
    else:
        plt.show()
    plt.close()

def plot_routes(best_solution, customers, depots):
    # Combine routes for each depot
    depot_routes = {}
    for depot in depots:
        depot_routes[depot['index']] = [route for route in
best_solution if route[0] == depot['index']]

    # Add the depot positions to the pos dictionary
    pos = {customer['index']: (customer['x'], customer['y']) for
customer in customers}
    names = {customer['index']: customer['index'] for customer in
customers}
    for depot in depots:
        pos[depot['index']] = (depot['x'], depot['y'])
    # Loop over depots and create separate figures for each route
    for depot_idx, depot in enumerate(depots):
        vehicle_routes = [route for route in
depot_routes[depot['index']] if len(route) > 2]
        for i, route in enumerate(vehicle_routes):
            draw_route(pos, route, names, i + 1, save_file=True,
file_name=f"Depot_{depot['index']}_Route_{i + 1}")

# Genetic Algorithm
def genetic_algorithm(instance_file, population_size,
num_generations, tournament_size, mutation_probability):
    # Step 1: Load instance data
    num_vehicles, num_customers, num_depots, customers, depots,
vehicles = load_instance_data(instance_file)

    # Step 2: Initialize the population
```

```
    population = generate_initial_population(population_size,
num_vehicles, num_customers, customers, depots, vehicles)

    # Step 3: Evaluate the fitness of the initial population
    fitness_values = []
    for individual in population:
        fitness_values.append(calculate_fitness(individual))
    generation_fitness = []
    best_solution_value = float('inf')
    best_solution = None
    initial_solution = bestIndividual(population)
    initial_solution_fitness =
calculate_fitness(initial_solution)
    # Step 4: Iterate through generations
    for generation in range(num_generations):
        # Step 4.1: Perform tournament selection to select
parents
        offspring = []
        new_population = []
        for _ in range(population_size):
            parent1 = tournament_selection(population,
fitness_values, tournament_size,threshold)
            parent2 = tournament_selection(population,
fitness_values, tournament_size,threshold)
        # Step 4.2: Create the offspring using best cost route
crossover
            if random.uniform(0, 1) < crossover_rate:
                child1, child2 =
best_cost_route_crossover(parent1, parent2)
                if child1 is not None:
                    offspring.append(child1)
                if child2 is not None:
                    offspring.append(child2)
        # Step 4.3: Perform swap mutation on the offspring
            for individual in offspring:
                if random.uniform(0, 1) < mutation_rate:
                    new_individual = swap_mutation(individual)
                    new_population.append(new_individual)
        best_individual = bestIndividual(new_population)
        current_solution_fitness =
calculate_fitness(best_individual)
        if initial_solution_fitness < current_solution_fitness:
            current_solution_value = initial_solution_fitness
            best_individual = initial_solution
        else:
            current_solution_value = current_solution_fitness
        if current_solution_value < best_solution_value:
```

```
            best_solution_value = current_solution_value
            best_solution = best_individual
        generation_fitness.append(best_solution_value)
        # Step 4.4: Select best individual

    # Plot the best fitness value at each generation
    plt.plot(range(1, num_generations + 1), generation_fitness)
    plt.xlabel('Generation')
    plt.ylabel('Best Fitness Value')
    plt.title('Convergence of Genetic Algorithm')
    plt.show()
    best_distance =
sum(calculate_route_distance_duration(route)[0] for route in
best_solution)
    best_duration =
sum(calculate_route_distance_duration(route)[1] for route in
best_solution)
    return best_solution, best_distance, best_duration,
best_solution_value,generation_fitness


#individual = generate_random_individual()
instance_file = 'D:\\Thesis\\Project_thesis\\pr21a.txt'
num_vehicles, num_customers, num_depots, customers, depots,
vehicles = load_instance_data(instance_file)
distance_matrix,travel_time_matrix = generate_distance_matrix()
save_distance_matrix(distance_matrix, 'distance_matrix.csv')
save_travel_time_matrix(travel_time_matrix,'travel_time_matrix.cs
v')
best_Ind, best_distance, best_duration,
best_solution_value,generation_fitness =
genetic_algorithm(instance_file, population_size,
num_generations, tournament_size, mutation_rate)
# Calculate statistics over all generations
mean_fitness_values = np.mean(generation_fitness)
std_fitness_values = np.std(generation_fitness)
min_fitness_value = np.min(generation_fitness)
max_fitness_value = np.max(generation_fitness)
print((best_Ind))
print(best_distance)
print(best_duration)
print(best_solution_value)
print(f"Mean Fitness Value: {mean_fitness_values}")
print(f"Standard Deviation of Fitness Values:
{std_fitness_values}")
print(f"Minimum Fitness Value: {min_fitness_value}")
print(f"Maximum Fitness Value: {max_fitness_value}")
```