TU Clausthal

# A COMPARATIVE ANALYSIS OF META-HEURISTIC OPTIMISATION ALGORITHMS FOR VEHICLE ROUTING

BACHELOR'S THESIS

presented by

DESPINA TAWADROS

Mobility and Enterprise Computing group
Department of Informatics
Technische Universität Clausthal

Despina Tawadros: *A Comparative Analysis of Meta-heuristic Optimisation Algorithms for Vehicle Routing*

STUDENT ID
487115

Email: despinatawadros@hotmail.com

ASSESSORS
First assessor: Dr. Jelena Fiosina
Second assessor: Prof. Dr. Jörg P. Müller

DATE OF SUBMISSION
March 15, 2022

## ABSTRACT

Today's popular "shared-economy" concept has accelerated the progress towards prioritising the implementation of cost-efficient strategies and promoting sustainable solutions, especially in the logistics and transportation sectors. This has directed more attention towards the last-mile delivery logistics and the prohibitively expensive costs associated with it. Hence, numerous approaches tackle this challenge by utilising urban transportation resources in the supply chain. This is achieved by incorporating parcel delivery into ride-sharing services by integrating parcel delivery requests with passenger transportation route plans. Lying at the core of route planning is the travelling salesman problem, which is a typical unresolved challenge in the field of combinatorial optimisation, owing to the exponential time complexity and expensive resources required by traditional algorithmic methods to compute the optimal route. Meanwhile, several heuristic and meta-heuristic optimisation techniques such as evolutionary algorithms, swarm intelligence-based algorithms or physics-based algorithms have been developed in recent years in an attempt to effectively solve optimisation problems. This thesis aims to compare the performance of three different types of meta-heuristic algorithms (Genetic Algorithm, Ant Colony Optimisation and Simulated Annealing) for the vehicle routing problem represented as travelling salesman problem, based on two datasets: NYC Green Taxi dataset and Porto City Taxi dataset. The performance of the three algorithms was evaluated on the basis of computation efficiency in terms of time needed to find the solution and on the basis of accuracy as well as reliability of the obtained solutions. The simulation results obtained from both datasets show that simulated annealing provided the most accurate and most reliable results, where ant colony optimisation came in second. On the other hand, the genetic algorithm demonstrated the highest efficiency in terms of algorithm runtime yet the quality of its results fell short in terms of accuracy and reliability.

# ZUSAMMENFASSUNG

Das heute beliebte „Shared-Economy“-Konzept hat den Fortschritt hin zur Priorisierung der Umsetzung kosteneffizienter Strategien und der Förderung nachhaltiger Lösungen beschleunigt, insbesondere im Logistik- und Transportsektor. Dies hat mehr Aufmerksamkeit auf das Problem der Zustellung auf der letzten Meile und die damit verbundenen unerschwinglich hohen Kosten gelenkt. Daher gehen zahlreiche Ansätze diese Herausforderung an, indem sie von städtischen Transportressourcen in die Lieferkette profitieren, indem sie die Paketzustellung in Ride-sharing integrieren, indem Paketzustellanfragen mit Routenplänen für die Personenbeförderung integriert werden. Im Mittelpunkt der Routenplanung steht das Problem des Handlungsreisenden, das aufgrund der exponentiellen Zeitkomplexität und der teuren Ressourcen, die herkömmliche algorithmische Methoden zur Berechnung der optimalen Route benötigen, eine typische ungelöste Herausforderung im Bereich der kombinatorischen Optimierung darstellt. Inzwischen wurden in den letzten Jahren mehrere heuristische und metaheuristische Ansätze wie evolutionäre Algorithmen, Schwarmintelligenz-basierte Algorithmen oder physikbasierte Algorithmen entwickelt, um Optimierungsprobleme effektiv zu lösen. Diese Arbeit zielt darauf ab, die Leistung von drei verschiedenen Arten von meta-heuristischen Algorithmen (genetischer Algorithmus, Ameisenalgorithmus und simuliertes Annealing) für das als Problem des Handlungsreisenden dargestellte Vehicle Routing Problem zu vergleichen, basierend auf zwei Datensätzen: NYC Green Taxi Datensatz und Porto City Taxi-Datensatz. Die Ergebnisse wurden auf der Grundlage der Recheneffizienz in Bezug auf die zum Finden der Lösung benötigte Zeit und auf der Grundlage der Lösungsgenauigkeit sowie der Zuverlässigkeit der erhaltenen Lösungen bewertet. Die aus beiden Datensätzen gewonnenen Simulationsergebnisse zeigen, dass Simulated Annealing die genauesten und zuverlässigsten Ergebnisse liefert, wobei die Optimierung der Ameisenalgorithmus an zweiter Stelle stand. Andererseits zeigte der genetische Algorithmus die höchste Effizienz in Bezug auf die Laufzeit, die Ergebnisse blieben jedoch in Bezug auf Genauigkeit und Zuverlässigkeit zurück.

## EIDESSTATTLICHE VERSICHERUNG

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, wurden als solche kenntlich gemacht. Die Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsstelle vorgelegt.

*Clausthal-Zellerfeld, den 15. März 2022*

Despina Tawadros

## PUBLICATION AGREEMENT

Bereitstellung in der Bibliothek Ich erkläre mich nicht mit der öffentlichen Bereitstellung meiner Bachelor's Thesis in der Instituts- und/oder Universitätsbibliothek einverstanden.

*Clausthal-Zellerfeld, den 15. März 2022*

Despina Tawadros

# ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Dr. Jelena Fiosina for her continuous support. Her invaluable guidance and motivation have helped me improve throughout the whole process. I am very grateful for what she has offered me.

I would also like to extend my gratitude to Prof. Jörg P. Müller, for giving me the chance to join his research group and leading me to work on this exciting topic.

And to my parents, and their unfailing support and continuous encouragement from my very first step in this journey.

Most of all, to God almighty whose endless blessings guided every step I took throughout those years.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ACRONYMS

**VRP** Vehicle Routing Problem

**TSP** Travelling Salesman Problem

**GDP** Gross Domestic Product

**CVRP** Capacitated Vehicle Routing Problem

**VRPPDP** Vehicle Routing Problem with Pickup and Delivery

**VRPTW** Vehicle Routing Problem with Time Windows

**GA** Genetic algorithm

**SA** Simulated Annealing

**ACO** Ant Colony Optimisation

**RMSLE** Root Mean Sqared Logarithnic Error

**MAE** Mean Absolute Error

# 1 INTRODUCTION

## 1.1 MOTIVATION

Sustainability and efficiency are becoming key instruments regarding modern industrial development in various sectors. Particular attention is paid to the logistics and transportation sectors as they are going through a significant evolution aiming to fulfil an eco- and cost-efficient future using various strategies such as focusing on reducing societal energy use, resources, and expenses. As a result, the challenge of efficient vehicle routing has become a major concern for any supply chain, especially when dealing with last-mile logistics.

Last-mile logistics encompass the final step in the shipping process, where freight is delivered from distribution centres to customers [Bos20]. Due to the lack of scale effect, route planning for the last mile is the most expensive, resource-intensive and can be the most inefficient part of the entire supply chain [Ran+18]. The attempt to minimise the costs and resources in last-mile logistics lies at the core of the vehicle routing problem as inaccurate route planning is associated with driving extra miles, wasting time, increasing the carbon footprint and missing delivery times [Des+20]. Moreover, carriers must also be prepared to deal with a level of unpredictability during the delivery process which could affect the accuracy of the estimated delivery times. This obstacle occurs frequently, especially in urban areas with high population density and a higher probability of traffic congestion.

## 1.2 AIM AND SCOPE

In the light of the above discussion, we tend to alleviate the inaccuracy and inefficiency issues that relate to the process of static route planning which are indirectly associated with the prime obstacles in taking the first steps towards fulfilling the goals of sustainability and cost-efficiency. To start looking for solutions, we first need to bring the Travelling Salesman Problem (TSP) to the front line. The TSP is a specification of the Vehicle Routing Problem (VRP) which aims to find a Hamiltonian cycle with the minimum cost between a set of dispersed geographical points while passing through each point exactly once with an overall shortest trip cost. The TSP is a relatively complex problem that belongs to NP-hard optimisation problems because it cannot be solved in a polynomial-time [Pot96]. Traditionally, in order to find the shortest trip cost, all route permutations are first calculated then compared to determine the route with the least cost. The time taken to solve this problem increases exponentially as the number of inputs increases since the number of permutations increases at a factorial rate [Gup13]. As a result, many route optimisation algorithms have been recently proposed to calculate a near-optimal solution inasmuch as the balance between calculating the optimal route in a relatively short time is nearly impossible to find. Hence, by settling

for a near-optimal solution, a fraction of the solution quality is traded for calculation speed to accomplish the efficiency longed for [Des+15]. For this reason, we investigated and employed meta-heuristic algorithms in our thesis as an alternate method to solve the TSP in order to find a near-optimal and scalable solution. Meta-heuristic algorithms most of the time give a near-optimal result but in a relatively short time, if their parameters are correctly tuned [Doe+19]. Therefore, a comparative analysis of the time needed by each meta-heuristic to solve the problem and the accuracy of the obtained results was carried out in this thesis.

This thesis aims to compare the performance of three different types of meta-heuristic algorithms for the VRP where a list of drop-off or pick-up locations of a delivery trip is given. The performance of the algorithms are compared on the basis of finding a near-optimal trip route with the least possible margin of error (shortest travelling time) and fastest runtime. These algorithms are the Genetic algorithm (GA) which belongs to the evolutionary algorithms, Ant Colony Optimisation (ACO), a swarm intelligence-based algorithm and finally, Simulated Annealing (SA), a type of physics-based algorithm. The algorithms were tested on two different datasets: NYC Green Taxi dataset and Porto City Taxi dataset.

The following points sum up the objective of this thesis:

1. Provide an overview of state-of-the art strategies used to solve VRP in context of TSP and their contributions in parcel delivery route planning as well as in ride-sharing applications.
2. Examine the travelling salesman problem from a route optimisation perspective and its solution approaches, including traditional and meta-heuristic algorithms.
3. Analyse and pre-process appropriate datasets to conduct an experimental analysis
4. Analyse and employ Gradient Boosting - a machine learning-based method for estimating the travel time between different geographical nodes.
5. Search for a suitable implementation library for meta-heuristic algorithms, compare alternatives, adapt the implementation in the chosen library to TSP and apply it on the given problem statement.
6. Conduct experiments to compare the three mentioned meta-heuristics with the selected datasets by comparing the optimisation results of different optimisation algorithms and reach a conclusion over the best approach taking into account various evaluation criteria.

The optimisation algorithms were evaluated based on the following criteria:

1. Average computation time required to find the optimal path.
2. Increase in computation time over different problem sizes.
3. Computation efficiency by comparing number of iterations required for the three meta-heuristics to converge towards the optimal solution.
4. Solution accuracy and reliability by calculating the mean and standard deviation in 5 test runs.

## 1.3 OUTLINE OF THE THESIS

This thesis is structured as follows. The overall structure of the study takes the form of six chapters, including this introductory chapter. chapter 2 lays the theoretical groundwork of the thesis by providing a comprehensive insight into the TSP followed by an overview of some of the different approaches developed to solve this optimisation problem, followed by an in-depth look into the three different meta-heuristic algorithms. It concludes with discussing travel time prediction approaches from a theoretical aspect with a special focus on Gradient Boosted Trees algorithm and XGBoost library which was used in this thesis. chapter 3 describes briefly the steps taken for developing four-step a machine learning pipeline to solve the route optimisation problem starting from collecting and pre-processing the data till comparing and analysing the results. This chapter provides a summary to the practical steps taken towards fulfilling the aim of this thesis. Section 1 and Section 2 in chapter 4 provide a summarised descriptive analysis of the NYC Taxi and Porto Taxi datasets followed by a brief explanation of the pre-processing steps. The last two sections explain the travel time forecasting process through the implementation of a machine learning model using XGBoost to train the model and present the results obtained. chapter 5 covers implementation details of the three algorithms as well as the efficiency criteria to compare their results. Lastly, chapter 6 evaluates and compares the experimental results obtained from applying the three different algorithms on the two datasets from chapter 5. The conclusions in chapter 7 provide a summary of the thesis and the obtained results, and identifies areas for further research.

# 2

# BACKGROUND

## 2.1 VEHICLE ROUTING PROBLEM

Accurate planning and scheduling of rides are of significant importance to both, supply chain operations and ride-sharing applications [Kon20]. They play a key role in determining the overall trip costs and customer satisfaction which are reflected upon the revenues. Recently, sustainability too has become a key instrument in employing more efficient strategies to reduce the negative impact on the environment. The VRP is a classic combinatorial optimisation problem that first appeared in a [DR59] to address the distribution problem of freight delivery from a central warehouse to geographically dispersed customers and they proposed an algorithmic approach to solve this challenge. This approach was later more generalised and applied to a wide range of transportation and logistic planning challenges such as ride-sharing and last-mile delivery where the objective is to find the optimal route under various real-life problem-related constraints.

Transportation usually accounts for 10% of the product cost [RCS16], whereas 41% of the overall supply chain costs are owed to last-mile delivery. Moreover, only 80% of the delivery costs are charged by retailers to balance between their profitability and customer satisfaction [Sog]. Using optimisation programs in solving this issue can reduce the loss for companies by 5% and this is a considerable value, since the transportation sector constitutes 10% of the EU's Gross Domestic Product (GDP) [AGL07].

Additionally, urban transportation has been going through a rapid evolution, with the emergence of new concepts like the "shared-economy". With the recent growing interest in ride-sharing, real-time ride-sharing companies such as Uber or Lyft have been gathering large amounts of data to improve the efficiency of their transportation systems [BJM19]. It is argued in [Pil+13] that taxi routing has received limited attention in the field of VRP where the real problem sizes are large and last-minute requests have limited space for optimisation available.

With the explosive growth in e-commerce, package delivery demands have exceeded the last-mile delivery logistical capabilities, placing more pressure on courier services to maintain the same quality of service [WZL18] thus, directing less attention towards investing in sustainable solutions. However, today's popular "shared-economy" concept can provide a new alternative that takes advantage of ride-sharing in alleviating some of the pressure placed on the last-mile through bridging the gap between the massive demands and limited capabilities by making the most effective use of transportation sources [COCS20] while meeting sustainable planning goals by integrating ride-sharing with the logistical process.

Static and dynamic scheduling are two main classifications of the VRP. Static VRP is known as the offline variant as all of the requests are handled beforehand and are incorporated into the planned routing schedule [NKP12]. Dynamic VRP deals with real-time requests and integrates these demands into the route plan. However, periodically updating the offline or static solution as new requests become available, is also considered a dynamic approach [BJM19].

Central to the entire discipline of VRP is optimal route planning. VRP handles the goal of finding the optimal route to be travelled to visit a set of locations by multiple vehicles. The optimal route encompasses trips with the minimum travel distance, time, fuel costs or any weighed constraint such as vehicles with limited capacities for deliveries or visiting locations within a pre-specified time window [Bac+00]. The basic VRP handles solving the routing problem with one constraint and a single vehicle to find the optimal route where the starting and the ending location are the same. Constraints originating from real-life challenges faced by transportation and logistics corporations are transformed into VRP variants. The VRP is NP-hard, hence the size of the problem determines whether it can be solved optimally using combinatorial optimisation or not [Cor+07].

**Problem Setup:** VRP can be mathematically formulated as follows: $G = (V, A)$ is a complete graph, where $V = 0, 1, 2, ...., n$ is the set of nodes or customer locations and $A = \{(i, j)|i, j \in V, i \neq j\}$ is the set of edges connecting the nodes. $V_0$ denotes the depot or dispatching location. $X_{(i,j)}$ denotes the weight of the constraint for the edge between node $i$ and node $j$. Figure 2.1 illustrates the VRP where three vehicles are assigned different route plans starting from a central location to cover all the requests while minimising the overall travelling distance. Numbers on arrows represent path weights between each location pair.

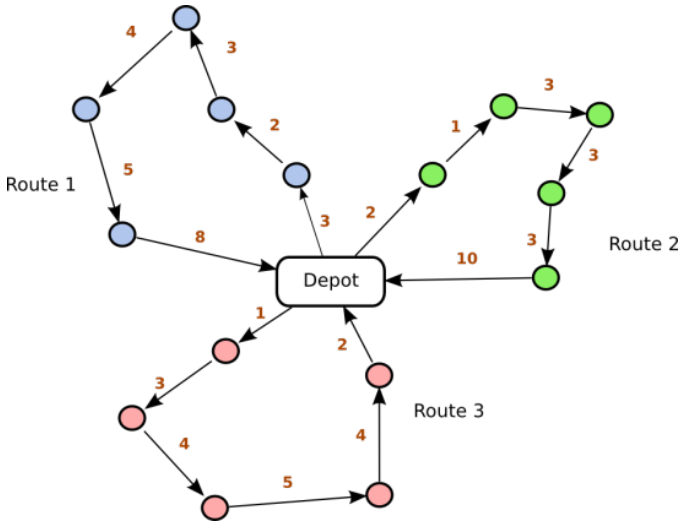



Figure 2.1: VRP Route Planing [1]

1 https://logisticsmgepsupv.wordpress.com/2014/04/10/vehicle-routing-problem-its-there-a-perfect-solution/

**VRP Variants:**

- **Capacitated Vehicle Routing Problem (CVRP):** This variant handles vehicles with limited carrying capacities such as weight or volume while taking the route with the least cost. This is one of the simplest and the most widely studied variant [Kon20] as it is often combined with other variants such as the time window constraint.
- **Vehicle Routing Problem with Pickup and Delivery (VRPPDP):** This variant considers picking up and dropping off parcels or passengers [Ber+07]. The pickup and delivery problem is classified into three sub-problems: one-to-one, one-to-many-to-one and many-to-one [BCL10]. Ride-sharing is modelled as a one-to-one which is usually referred to as the dial-a-ride problem for transporting people [Cor+03], while parcel delivery can be modelled as one-to-many-to-one where parcels are delivered from depot to many customer locations and parcels are picked up from customers and delivered back to the depot [VLM20].
- **Vehicle Routing Problem with Time Windows (VRPTW):** This variant considers picking up and dropping off parcels or passengers [Ber+07]. The pickup and delivery problem is classified into three sub-problems: one-to-one, one-to-many-to-one and many-to-one [BCL10]. Ride-sharing is modelled as a one-to-one which is usually referred to as the dial-a-ride problem for transporting people [Cor+03], while parcel delivery can be modelled as one-to-many-to-one where parcels are delivered from depot to many customer locations and parcels are picked up from customers and delivered back to the depot [VLM20].

**VRP Applications**

- **Last-Mile Logistics:** In last-mile logistics, vehicles transport parcels from distribution centres to customers where the delivery schedules are planned in advance. Customers are also associated with a delivery time window in case of any unplanned delays take place. This is a static variant of the VRP as the order of locations is planned prior to the delivery process. Heuristics are usually the classical approach to solve the static vehicle routing problems, several heuristics for VRP with time windows are discussed in the survey [BG05].
- **Ride-Sharing:** In contrast to the traditional ride-hailing service where a transportation service handles a single ride request at a time, in ride-sharing, multiple parallel ride requests share a single vehicle [Yue+19]. The major route planning challenges are building trust among unknown passengers and assigning compatible ride orders to the same vehicle without taking excessive detours resulting in much longer trip durations. A threshold of extra time for detour needs to be determined as well as a maximum number of passengers. The objective is to reduce traffic congestion and travel costs, and to maximise the profit by increasing the number of passengers [Ren+20].
- **Freight Integrated Transportation:** Recent studies have been made to combine ride-sharing with parcel delivery, in order to maximise efficiency. For ride-sharing with parcel delivery we refer to [Li+14] and [Li+15] which address the challenge of planning fixed trip routes for taking all passenger requests beforehand and accommodating as many parcels as possible within specific constraints related to time window and vehicle capacity. They propose two mathematical models for formulating a static plan for passenger and parcel requests that were both known

earlier. The second model formulates the insertion of parcel requests within a given passenger transportation route.

## 2.2 TRAVELING SALESMAN PROBLEM

The Traveling Salesman Problem TSP is a classical problem that is extensively studied in the field of combinatorial optimisation. It aims to find the minimum trip cost for example in terms of minimum distance or shortest time through a set of N nodes, where each node is visited exactly once [Pot96]. Furthermore, the TSP has applications in various areas such as in logistics [FO11], internet planning and DNA sequencing [MPG12]. The TSP is simple to state, yet difficult to solve, this is due to the fact that it belongs to NP-hard problems and therefore cannot be solved in polynomial time [Pot96]. With applying the brute-force algorithm, the computational complexity and the time needed to find the optimal solution grow exponentially as the number of nodes increases O(n!). Brute force algorithm calculates and compares all possible permutations of paths between all the nodes to determine the shortest route. This results in a factorial increase, which imposes a scalability problem as beyond a specific number of nodes the solution cannot be calculated in real-time [Kol15]. This thesis considers the simplest case where the challenge of searching for a route with the shortest trip duration with one vehicle is tackled through conducting a comparative analysis on three different meta-heuristics applied to solve the TSP. This approach in general lays the groundwork for extending the basic TSP to a variety of vehicle routing problems.

The problem is mathematically stated as follows:

$$\text{Minimise} \sum_{(i,j)\in L} c_{ij}x_{ij},\ x_{ij} \in \{0,1\}$$

$$\text{Subject to} \sum_{(i,j)\in L} c_{ij}x_{ij} + \sum_{(i,j)\in L} c_{ji}x_{ij}$$

$$\text{With the subtour breaking constraint} \sum_{(i,j)\in L:\{i,j\}\in S} x_{ij} \leqslant |S|-1, \forall S \subset N : 2 \leqslant |S| \leqslant N-2$$

where the TSP is defined on $N = \{1,2,..,n\}$ nodes, $x_{ij}$ identifies whether the path between the nodes $i$ and $j$ exists (1 if exists and 0 otherwise). $c_{ij}$ denotes the weigh of the path $x_{ij}$ and $S$ is the set of all tours.

However, a near-optimal solution can be obtained in polynomial time if we trade solution quality for calculation speed. Prioritising the need to find good solutions in a relatively short time and cutting down computation costs has led to the development of approximation algorithms that tend to explore a limited but promising sector of the solution space with high efficiency. Hence, we settle with a near-optimal solution with a shorter time. Approximation algorithms [TY21] can be classified as illustrated in Figure 2.2.

### 2.2.1 *Optimisation heuristics*

As we mentioned above, classical exact methods to find the optimal solution are exhaustive and impractical. Heuristic methods fall under the approximation algorithms category as they speed up the process and settle with satisfactory solutions [MSM10]. They can be further classified into the following categories:

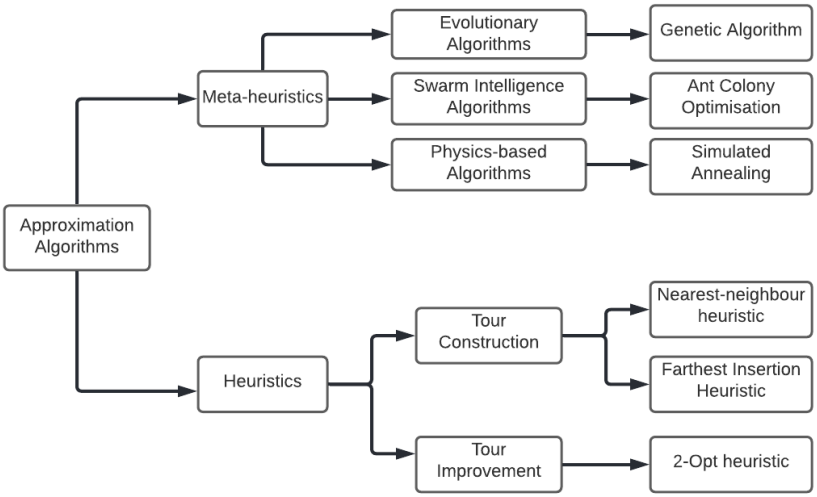



Figure 2.2: Classification of approximation algorithms

1. **Tour Construction Algorithms**

   These algorithms focus only on constructing a tour according to some rules apart from the optimality of the final result. Parts of the tour are constructed step by step successively without making any change or further improvements. They're greedy in their nature as they follow a local optimal strategy by choosing the next path which results in the minimum increase in the tour cost [MSM10]. However, a locally optimal solution does not always yield globally optimal result because it depends on previously made choices without considering future consequences and choices, and its inability to reverse them. Nearest-Neighbour Heuristic with complexity $O(n^2)$ works by choosing a random node then successively adding new nodes with the shortest distance to the last node added. It follows a greedy strategy where it makes decisions based on local limited information that guarantee a locally optimal solution. It starts by choosing the next shortest path to the next unvisited node at each construction step [MV17]. However, this strategy does not guarantee a globally optimal solution because the quality of the resulting tour depends heavily on the first randomly chosen node. Another algorithm is Farthest Insertion Heuristic with complexity $O(n^2)$. The farthest insertion algorithm also follows a greedy notion but exceeds the performance of the nearest-neighbour algorithm. It starts by forming a random sub-tour, usually consisting of only 3 edges joined together with 3 random nodes forming a triangle. It selects an outside edge with the longest distance to any of the vertices within the sub-tour and inserts this node such that the total length of the sub-tour is maximised. This process is then repeated iteratively [RSI77].

2. **Tour Improvement Algorithms**

   Tour improvement algorithms start with a randomly generated tour or with a tour generated using construction heuristics and continuously improve the tour by modifying some parts with every iteration until a near-optimal result is reached [MSM10]. One of such algorithms is 2-Opt Heuristic. This algorithm

focuses on continuously improving the resulting tour through reordering the edges crossing over each other and resulting in a longer tour. Crossing edges are randomly selected from either a randomly generated tour or a tour generated by one of the construction heuristics, and reconnected or swapped with each other if they lead to a shorter tour with each iteration [Nil03].

**Advantages of Heuristics:**

- ▷ Tour Construction algorithms are frequently employed to create an initial solution and are further optimised with meta-heuristics [Kon20].
- ▷ Can find good solutions in short time when the search are is restricted [ZL].
- ▷ They are practical and easy to implement for small-sized problem instances.
- ▷ Unlike exact methods, which are suitable only for small-sized problems, heuristics have no bounds on problem size and constraints as they can still find a near-optimal solution efficiently [BRN15].

**Disadvantages:**

- ▷ They lack flexibility as they are usually problem-specific and need to be adapted to the problem, especially when various constrains are considered [BRN15]. They require a good knowledge in the problem domain.
- ▷ Intermediate solutions are not allowed to deteriorate during the search process of finding better solution. Thus, convergence is not possible and often get trapped in local minima [Lap09].

## 2.3 META-HEURISTIC APPROACHES

They are strategies that guide the process of searching the solution space to find the near-optimal solution. They are stochastic search methods that encompass optimisation algorithms that try improving a candidate solution with each iteration to converge towards the optimal solution. Meta-heuristics encapsulate a learning-based strategy that combines tour construction with tour improvement heuristic methods while using past enhancements with each iteration to build neighbourhood search processes for exploration and exploitation [HCK12]. [HI17], [LQS20] and [MSM10] carried out an experimental analysis over the performance between several heuristics including Tabu search and meta-heuristics such as GA, ACO and SA in TSP. It was generally observed that meta-heuristics outperformed Tabu search especially in terms of solution accuracy with large size of input nodes and in the overall average computation time.

However, it is important to highlight the absence of a dominating approach and a stable performance over all experimental cases, since meta-heuristics are not used to find solutions for hypothetical problems, rather empirical problems [ASK17]. Therefore, the varying problem conditions can result in some degree of variation in the overall performance on each problem set. This is observed in the results obtained from the conducted experiments on the datasets of both cities, NYC and Porto in later sections.

**Comparison of Meta-heuristics and Heuristics**

- ▷ Meta-heuristics can considerably boost a system's computing capability without raising the hardware cost [ZL].
- ▷ Unlike heuristics, meta-heuristics are not problem-specific and therefore can be applied to various domains and solve a wide range of problems [BRN15].
- ▷ Meta-heuristics are non-deterministic, and they often produce sufficiently good results in solving NP-hard problems, especially with limited computation capacity, but they can never guarantee an optimal solution [ASK17].
- ▷ In contrast to heuristics, it is notable that meta-heuristics are flexible and can be adapted according to the requirements of the optimisation problem specifications to fit the prioritised goals on the basis of solution quality and computation time which vary across different optimisation scenarios. However, this comes at the cost of the notable effort required to design a problem-specific adaptation, since they lack a universally applicable design [BJM19].
- ▷ Due to the large solution space that requires searching for TSP, meta-heuristics outperform heuristics in solving TSP with large input instances in a reasonable time. In spite of their practical efficacy, guaranteeing a near-optimal solution relies on proper implementation and hyperparameter tuning, which in itself is another optimisation landscape [SG13].
- ▷ Meta-heuristics employ local and population search methods that help promote solution space exploration and exploitation which reduces the chances of getting stuck in local minima. On the other hand, heuristics are mainly local search methods and hence get stuck more often in local minima and this explains the difference in the quality of the results [Kon20].
- ▷ Meta-heuristics encompass mechanisms that prevent them from getting trapped in local minima by temporarily accepting worse solutions and performing a broader search of the solution space [NM16].

Moreover, machine learning-based problems rely heavily on large datasets, therefore, it is difficult to aim for optimality with the existing traditional approaches. They intelligently simulate different concepts for exploring and exploiting the search space and hence can be classified according to how they operate over the search space. They can be further classified into three categories:

- ▷ Evolutionary Algorithms
- ▷ Swarm Intelligence Algorithms
- ▷ Single-Point Algorithms

### 2.3.1 *Evolutionary Algorithms: Genetic Algorithm*

Evolutionary algorithms imitate the Darwinian theory of evolution. They are inspired by the analogy of survival of the fittest by replicating Darwin's notion of natural selection to find near-optimal solutions to real-world problems by making slight and slow gradual changes to the solution until the least-fit solution dies and another one

evolves and survives throughout the evolutionary whole process [Alt16].

**Related Works**: Recently, GA has caught a lot of attention from researchers that it became the second most applied meta-heuristic [Kon20]. GA was first proposed by Hohn H. Holland in [Hol84]. It is broadly applied to various VRP variants such as in [HCK12] where it was applied to solve dynamic VRP and in [NKP12] to solve VRP with time windows. Moreover, [Pot96] surveyed the application of GA with different operators on TSP from various papers and provided a short summary over its performance.

**Underlying Principles**: GA is the most popular algorithm among the evolutionary algorithms family. It is also referred to as a random-based classical evolutionary algorithm because it does not follow a specific analogy to find a solution as random changes are applied to the current solution to generate a new one [SB10a]. These changes are slow and gradual and tend to converge towards the optimal solution after several iterations.

**Convergence Strategy**: Natural selection [DW11] is the process of randomly selecting individuals within a population for mating and passing down their characteristics to their offsprings. However, based on the degree to which these offsprings are considered to be fit and able to adapt in their environments, their chance of survival is decided. Over time, individuals within the population who could not adapt fail to survive and die out, leaving out the most fit individuals to form new generations that consistently improve over time. Hence, they produce offsprings having the superior characteristics that helped their parents survive. The concept of survival of the fittest is central to the entire discipline of natural selection. Besides inheriting the characteristics from their parent, the offsprings mutate and have additional characteristics along with the inherited ones and these new characteristics can count as an overall improvement giving them better chances for survival and passing down these characteristics to a new generation and so on. The world that we live in is unlike many views, not static but dynamic with a constant change therefore evolution is a continuous process. However, for solving real-world problems we adopt the static view as the problem and the conditions are constant and do not change over time therefore this helps us to converge towards an optimal solution given the conditions.

**GA for TSP**: GA initially works over a population, which is the solution space having a number of individuals that correspond to the number of possible solutions [Gol+80], meaning that each individual is a potential solution. A fitness function is used to evaluate each individual within the population and to assign a fitness score. This fitness score indicates the quality of each individual or solution. As shown in Figure 2.3 each individual has a chromosome which is a set of features that define an individual. Genes form a chromosome. A gene is made up of an allele and a locus. An allele is a specific value, and a locus is its location on the chromosome [Mal17].

The process of GA is described in the following steps and illustrated in Figure 2.4 [GBC15].

1. *Initialization:* A population with a distinct number of individuals is first randomly generated.

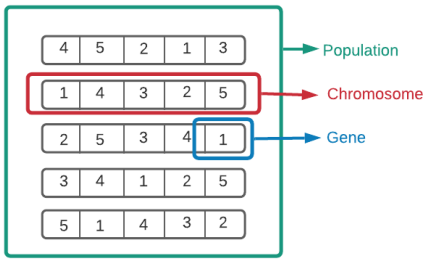



Figure 2.3: GA Elements

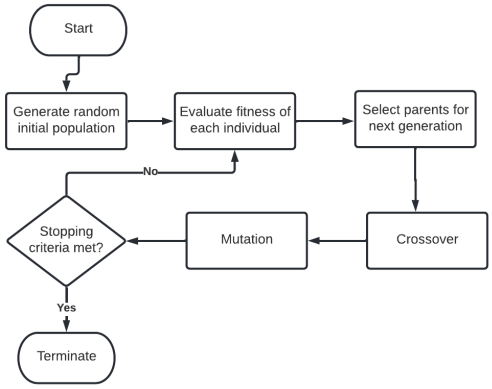



Figure 2.4: GA Flowchart

2. *Evaluation and Selection:* Each individual or solution is assigned a score through the fitness function. The higher the score of the solution the better the solution is. To simulate survival of the fittest, individuals with high fitness scores according to a pre-specified threshold are selected and added to the mating pool. Each two pairs of individuals are selected as parents for mating, either sequentially or randomly. Individuals who died out before reproduction are the ones with a low fitness score.
3. *Crossover:* After a pair of individuals is selected, the crossover process takes place as follows: a random point or position on each chromosome of the parents is randomly selected and genes on those chromosomes are swapped before and after this point. As a result, we end up with a new generation of offspring where their number of genes are identical to their parents' and their chromosomes are mixture of genes inherited from both parents. Without crossover, the new generation of offspring would be identical to their parents.
4. *Mutation:* If we solely depend on mating between parents with different genetic makeup, the algorithm will eventually get stuck in local optima without variations exploring other parts of the solution space. Therefore, to avoid this local convergence, mutation is responsible for making small random changes to random individuals such that we end up with a new generation with a wider gene pool. One way to do this is by selecting a random gene on the chromosome and changing its value. The degree of mutation in an offspring is governed by a probability distribution.
5. *Termination:* This process is repeated with steady pressure towards convergence until we reach some maximum number of generations or a threshold of performance, where we end up with a significantly improved gene pool of individuals with solutions close to the optimum.

### 2.3.2 *Swarm Intelligence Algorithms: Ant Colony Optimisation*

Swarm Intelligence algorithms are sometimes referred to as population-based algorithms since the collective behaviour of individuals influences the overall search behaviour. Individuals are autonomous agents that communicate with each other to find the best solution. Swarm-based algorithms include Ant Colony Optimisation (ACO), Bee Colony Optimisation and Particle Swarm Optimisation. These algorithms are graph-based and simulate the environment for TSP using a grid of nodes where each node represents a location and edges refer to connections between nodes. All these three algorithms were used in [Kue20] to solve the TSP, with ACO giving the overall best results.

**Related Works**: ACO was first introduced in [DG97] primarily for solving the TSP. It has been widely applied to differed VRP variants such as with time window in [GTA99] using multiple colonies to successively optimise the result from the objective function by minimising the number of vehicles then the distance travelled. ACO is frequently compared to GA as in [SB10b], where the case of path planning, which is closely related to VRP, was addressed. The objective was to find the optimal path for a robot to travel where the environment is represented by a grid of cells and the results have shown that ACO outperformed GA in terms of result accuracy and computation time. Moreover, [LY13] proposed a hybrid meta-heuristic combining ACO with GA to solve the VRP with multiple-depots variant. GA was employed to optimise the parameters of ACO using cross and mutation operators in an attempt to reduce the effort in finding the optimal parameter value combinations which are very sensitive to change.

**Underlying Principles**: ACO is a swarm-based optimisation algorithm. It was first proposed by Marco Dorigo in [DBS06]. It employs bio-inspired concurrent and asynchronous agents moving incrementally through the states of a problem in search of adequate solutions to an optimisation problem [CT16]. These agents correspond to the cooperation of many autonomous local search processes each one building a complete solution [HCK12]. The algorithm emulates the behaviour of ants roaming around colonies searching for food and finding a path between their colonies and food sources. ACO is a probabilistic technique for solving a wide range of optimisation problems, starting with TSP. The algorithm adopts the behaviour of ants as they interact with each other and move in search of food. As ants explore new areas and search for food, they leave a trail pheromone which guides other ants in following the same path. This deposited pheromone level is a measure of the quality of the path that directs the search paths of the future ants [Ang+01].

**Convergence Strategy**: The main characteristic of artificial ants is their ability to use the information regarding the surrounding environment to adapt. The algorithm comprises two main phases: a tour construction phase and a tour improvement phase where the pheromone levels are continuously updated as the algorithm converges towards the optimum [Gao21]. First, assuming that the search space is a graph and locations are nodes on this graph connected to edges that represent routes. Each ant starts from a randomly selected node, and moves along the edges with each construction step. While moving, the ant keeps a memory of the travelled path so as not

to violate the TSP rule by taking the same path again and visiting a location once more. The shortest path picked will have a stronger pheromone trail as the pheromone concentration is inversely proportional to the total length of a tour. Accordingly, as another ant is placed at a random location, it is more likely to pick the edge with higher pheromone levels [CT16]. This process is repeated until it converges towards the shortest tour with the overall highest pheromone concentration.

The algorithm is mathematically formulated as follows [LZ16]:
The pheromone level on an edge is

$$\Delta\tau_{i,j}^{k} = \begin{cases} \frac{1}{L_k} & \text{k-th ant travels from i to j} \\ 0 & \text{otherwise} \end{cases},$$

where $\Delta\tau_{i,j}^{k}$ denotes the increase in the pheromone amount on the edge between nodes i and j, and $L^k$ is the total length of the tour found by the k-th ant. The concentration of pheromone on an edge deposited by the k-th ant is the inverse of the total tour length travelled by the same ant. We notice here that shorter tours get higher pheromone values than longer tours.

To calculate the amount of pheromone on each edge, we need to find the total amount of pheromone deposited by all ants that traversed this path and the evaporation rate of the pheromone denoted by a constant, $\rho$ .

$$\tau_{i,j}^{k} = \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} + (1-\rho)\tau_{i,j}.$$

Pheromone evaporation dictates the attractiveness of an edge for the ants thereby influencing their choices in taking the next path. Longer edges have higher evaporation rates as it takes longer for an ant to march over this path back again and dispose pheromone. By comparison, shortest paths will have higher pheromone density as they are marched over more frequently. Adding the evaporation rate in determining the pheromone level on an edge is essential in keeping the algorithm away from getting trapped in a local minimum. Without the evaporation rate, paths chosen by the first ants will be more attractive to the other ants which will in return restrict their exploration to only a limited part of the solution space [VMF04].

Ants select edges by applying a stochastic local decision strategy that considers all the edges directly connected to the ant's current position [VMF04]. The main factor determining the next edge to be taken is a probability that is decided by the pheromone level and the attractiveness of each edge. The attractiveness or desirability state transition $\eta_{i,j}$ is a measure of the proximity of cities, where $\eta_{i,j} = \frac{1}{L_{i,j}}$. It enables the inflow of general local information, independent of the pheromone distribution. $\alpha$ and $\beta$ are parameters that balance the influence of the pheromone level and desirability values respectively in assigning a probability value for the next edge to be selected.

$$P_{i,j} = \frac{(\tau_{i,j})^{\alpha}(\eta_{i,j})^{\beta}}{\sum((\tau_{i,j})^{\alpha}(\eta_{i,j})^{\beta})}.$$

**ACO for TSP**: Figure 2.5 provides a simplified visual representation of the steps taken to solve the TSP through applying the ACO method. Given two ants on a graph with 4 nodes A, B, C and D, and edges with different distances between the nodes. The

goal is visit these 4 nodes and return to the first one by traversing the graph. Given the following initial values: $\tau_0 = 1$, $\rho = 0.5$, two ants.

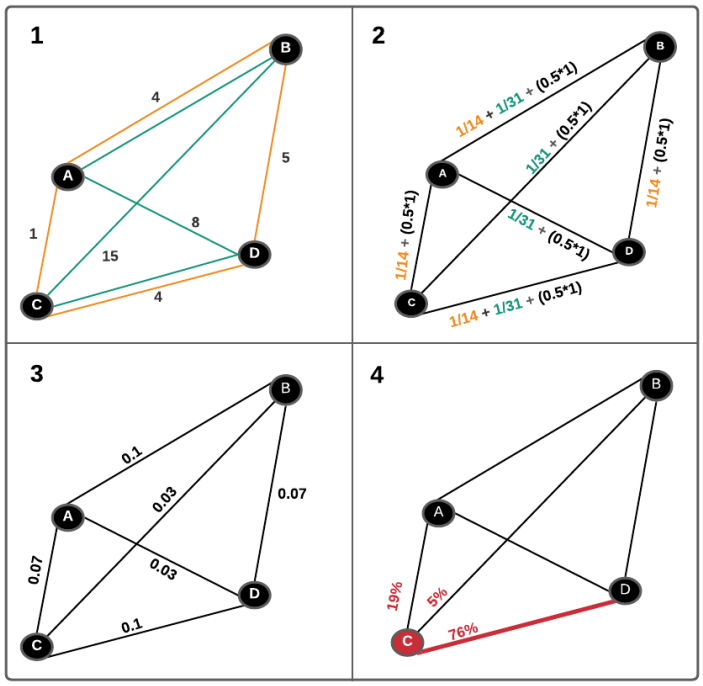



Figure 2.5: Applying ACO to TSP

In Figure 2.5 the first diagram, the orange route A-B-D-C-A is travelled by the first ant and the green route A-B-C-D-A by the second ant. The second diagram illustrates the pheromone concentration deposited by each ant plus the evaporation rate on each edge. In the third diagram the global pheromone value on each edge is displayed on each edge. In the next round if a new ant starts from node C there is a probability of 76% to select the path to node D. Figure 2.6 provides a summarised representation of the algorithm.

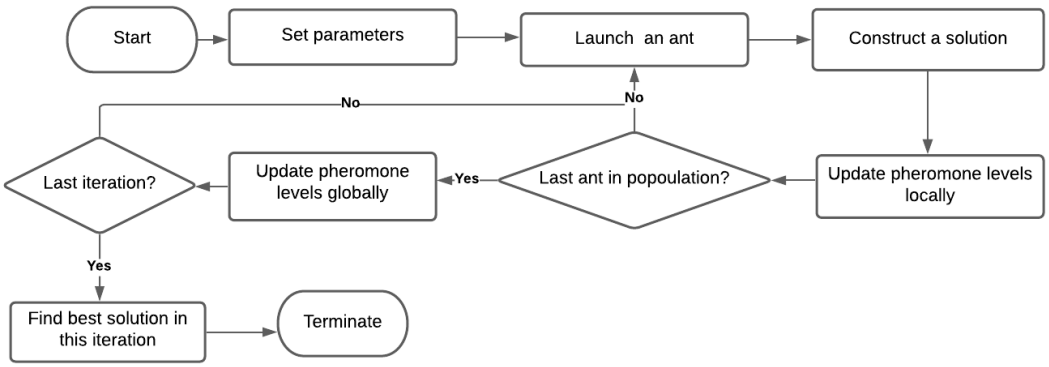



Figure 2.6: ACO Flowchart

### 2.3.3 *Single-Point Algorithms: Simulated Annealing*

Single-point or trajectory-based algorithms are search methods that operate on a single point in the solution space and iteratively improve the solution by exploring its neighbourhood for better solutions [RSF13].

**Underlying Principles**: SA as its name implies, exploits the analogy of the annealing process of the materials involving rapid heating then gradual controlled cooling of a material to change it to the desired structure. Heating weakens the molecular bonding within the material, supplies the molecules with more energy to move and makes the material ductile and easier to reshape. As the material gradually cools down, the harder it gets to change its shape. This strategy uses the cooling process for transforming poor unordered solution to an ordered satisfiable solution gradually [Bay+13].

**Related Works**: SA was first developed by [Met+53] to model the annealing process of solids, as particles arrange themselves to reach the thermal equilibrium and take shape. It was later introduced as a strategy to solve combinatorial optimisation problems by [KGV83]. SA is one of the first meta-heuristics proposed to solve the VRP [Kon20]. Moreover, [Lin+12] applies SA to a routing optimisation model to minimise operational costs for ride-sharing with multiple vehicles and pre-specified time windows. The performance of SA is frequently compared to GA in solving the ACO such as in [Kar+16] where the objective function aims to let experts visit the highest number possible of telecommunication companies distributed over many cities by finding the shortest path between the locations, and hence incorporating more companies to visit during their daily working hours. This study has shown that the solution quality of SA exceeded GA in terms of cost and run time. On the other hand, [Ade+12] compared the application of SA and GA to solve the TSP and yield an optimal solution. The results obtained from this study showed that GA slightly outperformed SA when searching for the shortest travel distance, while SA needed in comparison to GA shorter time to obtain the results.

**Convergence Strategy**: The high temperature in the algorithms is equivalent to the measure of randomness by which changes are made to the path while exploring the solution space. Lowering the temperature gradually narrows the search range until the algorithm converges towards the global optimum and the objective function decreases. In other words, high temperature increases the probability to accept worse solutions, allowing the algorithm to explore a wider space with sub-optimal paths to avoid the risk of becoming trapped in local optimum and increase the probability to find the global optimum [Bay+13]. Those changes are accepted with probability $p$ as denoted here [Wan+09]:

$$p = e^{-\Delta c/t}$$

where $\Delta c$ is a change in the objective function and $t$ is the current temperature. This process makes the algorithm very suitable to find an optimum within a large solution space. The following factors control the performance of the algorithm:

- ▷ *Cooling Schedule*: A maximum temperature as the initial temperature and a cooling rate until minimum temperate is reaches which is an optional termination condition.
- ▷ *Cooling rate*: The rate at which the maximum temperature decreases till it reaches the minimum temperature.
- ▷ *Move set*: Strategies to generate new solutions, such as swapping or reversing elements.

While the majority of the parameters are fixed values, the performance of the algorithm relies heavily on the parameters of the cooling schedule therefore, they need to be carefully selected. As stated before, SA explores the solution space by generating new solutions. Figure 2.7 below is a representation of generating and accepting a new neighbouring solution through the swapping process, despite being less optimal. Elements in the second and fourth positions swapped together to generate a new solution with a new order and a longer overall path.

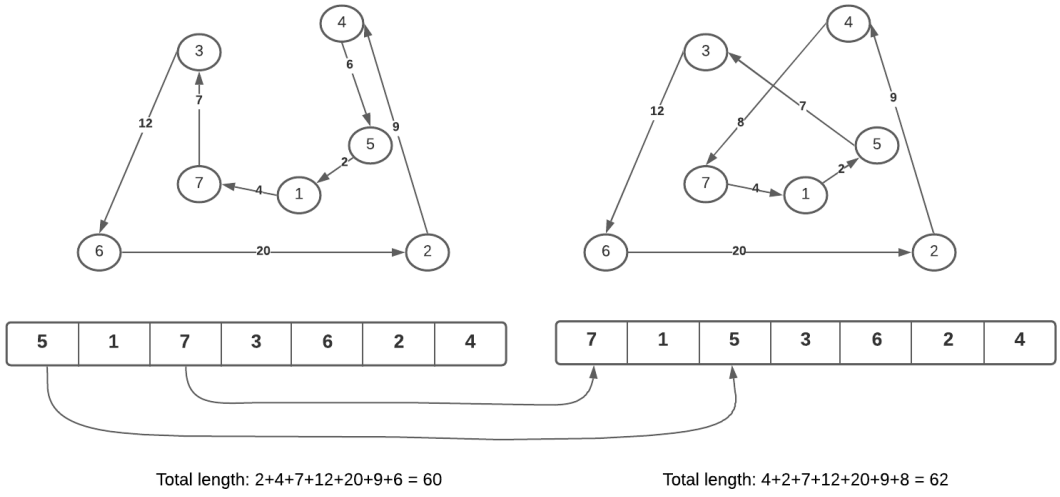



Figure 2.7: Swapping Operator in SA

**SA for TSP**: Applying the SA to TSP can be summed up in the following steps [Gup13] and visualised in Figure 2.8:

1. Create an initial solution in form of a list of locations to be visited once but in a random order.
2. Select a maximum temperature as initial temperature and define a cooling rate.
3. Starting at the initial temperature, loop through $n$ iterations. At each iteration, select 2 random locations and swap them.
4. Calculate the cost of the new tour. If the cost of the new tour is less, then accept the solution, else accept the tour (with the greater distance) under a certain probability.
5. Reduce the temperature at every iteration according to the cooling factor.

6. Stop when maximum number of iterations is met, or minimum temperature is reached or when termination criteria is met.

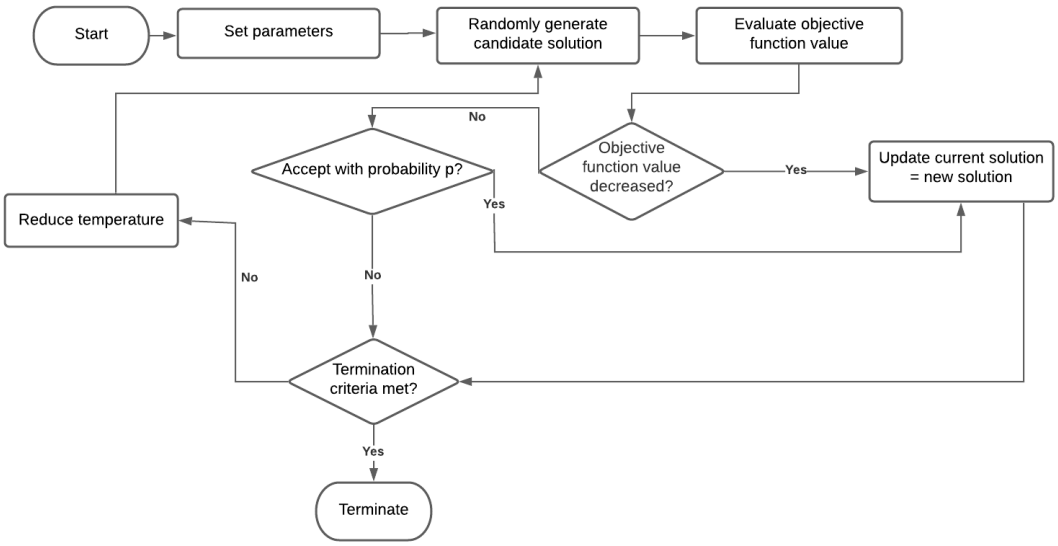



Figure 2.8: SA Flowchart

## 2.4 PROS AND CONS OF DIFFERENT META-HEURISTICS

| Algo. | Pros | Cons |
|---|---|---|
| GA | - Crossover and random mutation aid the algorithm in exploring wider areas in the solution space.<br>- Converges in a relatively short | - Gets very often trapped in local minima (premature convergence) |
| ACO | - A very efficient mechanism since throughout the process only partial solutions are constructed but still a large search space is covered.<br>- Able to collect local information that aid in the search process in the form of local heuristics, which reduce the marginal errors that result from using a pure probabilistic approach. | - Restricted to optimisation problems that can only be represented as graphs on which solutions can be constructed. |
| SA | - Performance of the algorithm relies heavily on the parameters of the cooling schedule therefore, they need to be carefully selected and tuned.<br>- Quite versatile and is not restricted to a specific problem domain. | - Selecting the correct parameters is a challenge of its own and can lead to inadequate solutions with expensive computation time if not correctly selected.<br>- Requires a high run time especially when the annealing schedule is long to run sufficiently enough iterations to make sure that the optimum is found. |

Table 2.1: Pros and Cons of the different meta-heuristics

## 2.5 TRAVEL TIME PREDICTION

According to [Zha+22] Travel time prediction refers to estimating the time needed to travel a distance between points based on given spatiotemporal data such as time of day and climate conditions as well as other data such as traffic conditions and passenger number. It became an effective measure in the past decades for its importance in implementing Intelligent Transportation Systems which pushes the urban design one step closer towards the long-sought concept of smart cities. However, the uncertainty of many external factors makes it challenging to execute scheduling for transportation and logistic operations such as with the cases of ride-sharing and last-mile delivery, resulting in low reliability and delays. Traditional prediction methods that are based on statistical models lack intelligence and adaptability especially when various factors are taken into consideration before planning [Zha+22]. The following data-driven approaches that utilise historical travel time data have been more frequently applied in travel time forecasting:

1. **Linear Regression:** It is the traditional model-based method, owing to its simplicity in implementation and interpretation. The target value is a linear function of the input independent variables [QF21]. These independent variables are typically traffic data gathered from past time intervals. It aims to minimise the error between predicted and actual value [Zha+22].
2. **Support Vector Regression:** It is a machine learning-based classification algorithm that predicts continuous variables [Wu+03]. It allows us to specify the tolerance of errors by defining a marginal error range (hyperplane). Points lying within this threshold arr of least importance, while data points outside the hyperplane are called support vectors which are used to plot the line showing the predicted values of the algorithm in an attempt to fit the best line within the threshold value [Bak20]. [Wu+03] compared Support Vector Regression to other baseline predictors: Current-time predictor and Historical-mean predictor in forecasting travel time using traffic data. Their results have shown that Support Vector Regression had overall better predicted values and much less Root Mean Error and Root Mean Squared Error values than the other methods.
3. **Neural Networks:** The non-linear modelling capability of neural networks has made them widely used. Neural networks are statistical models that are based on real-world systems developed by parameter tuning. The training process summons tuning the correct parametric values to minimise the error between the predicted and the actual output values [LZ05]. They demonstrate good performance over problem-specific applications but they lack generalisability [Zha+22]. [AH20] adopt two different neural network variants to predict travel time between bus stops. Both models showed an overall good performance with near accurate predictions.
4. **Gradient Boosting:** It is a machine learning-based approach that uses a set of decision trees to continuously improve the prediction strength of a model [Kan+19]. It works by combining several base models sequentially manner to create a model with a better performance. [Gup+18] employed gradient boosting and Random Forrest models on Porto taxi dataset to compare the performance in predicting travel time and the results showed that the gradient boosting model outperformed Random Forrest by giving better prediction results.

### XGBoost

In order to proceed with testing the performance of the mentioned meta-heuristics on NYC and Porto Taxi datasets to plan a route with the shortest travel time possible, we first need to predict the trip duration between different location pairs. The gradient boosting approach was first proposed by Chen and Guestrin in [CG16] and is adopted using XGBoost library to forecast the travel time between two locations within our datasets. XGBoost, which stands for eXtreme Gradient Boosting, implements machine learning algorithms under the gradient boosting framework designed for fast computation speed and high accuracy [CF21]. It is widely used in data mining and machine learning challenges as it is well known in solving complex prediction problems such as in [ZZ17]. XGBoost generally adopts the idea of additive training, where at each iteration new models predicting residuals between target values and current predicted values of existing models are created, which then are added to the prior models sequentially to make the final prediction until no improvements are observed [CG16]. "Gradient boosting" refers here to the usage of gradient descent algorithms to reduce the loss when new models are added. In addition, XGBoost models scale well with fewer resource requirements and speed up the learning process through processing large amounts of data in a parallel way efficiently [Bro21] [CG16]. Moreover, traditional gradient boosting is prone to overfitting, XGBoost overcomes this obstacle by incorporating a regularization framework (regularized objective model) [Kan+19]. In [Kan+19], XGBoost was applied on NYC Taxi dataset to predict travel time and compared the predicted results obtained with benchmark prediction models which are Support Vector Regression and Neural Networks. The results showed that XGBoost is capable of reducing errors between predicted and actual values more than the other two models for short as well as for long travel distances.

**Accuracy Metrics**:

The following metrics were used to evaluate the performance of the model:

1. Root Mean Sqared Logarithnic Error (RMSLE) measures the ratio of predicted and actual trip duration. It is suitable for our model as we do not prioritise penalising large differences since the predicted and actual values can be large. Therefore, we consider error percentage instead of the absolute value of the error, meaning that the scale of the error is not critical. Furthermore, with RMSLE, underestimated trip durations are penalised more than overestimated trip durations.

$$\text{RMSLE} = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(\log(\text{predicted}_i + 1) - \log(\text{actual}_i + 1))^2} \tag{2.1}$$

2. Mean Absolute Error (MAE) provides a measure for the average for the absolute values of the errors.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\text{actual}_i - \text{predicted}_i| \tag{2.2}$$

# 3 SYSTEM DESIGN

This chapter briefly describes the steps of a machine learning pipeline (Figure 3.1), developed to implement the necessary experiments using real-world datasets to compare different results obtained by the three meta-heuristics. As mentioned before, we focused on solving the previously stated problem formulated as TSP assuming that we have one vehicle and without considering any further problem-related constraints such as time window, vehicle capacity or detour time limit. The implementation steps are explained in more detail in the next chapters.

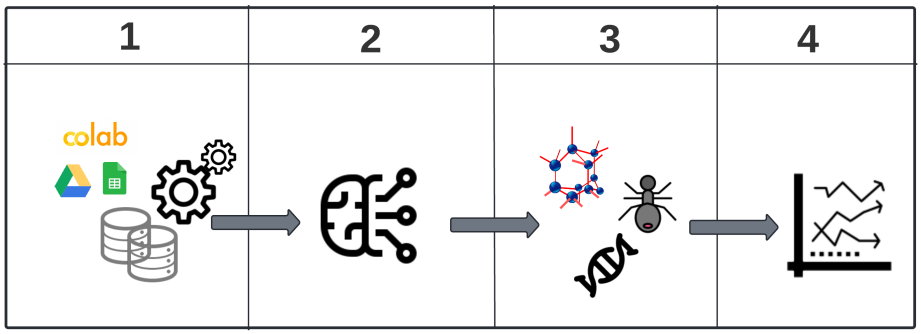



Figure 3.1: Implemented pipeline

## 3.1 PIPELINE DESCRIPTION

**Step 1: Data Analysis and Pre-processing**. This step covers preparing the necessary data for conducting the experiments. We worked with two datasets: NYC Green Taxi dataset and Porto City Taxi dataset. The datasets were saved as .csv on Google Drive and were then imported to Colab for pre-processing and further implementations. To prepare the datasets, we first pre-processed the data by removing outliers and performing feature engineering. Details on the datasets as well as the preparation process are found in chapter 4.

**Step 2: Travel Time Forecasting**. The goal of this step is to predict the travel time between any given pair of locations from the datasets. In order to achieve this goal, we first trained a model on each dataset using XGBoost after defining the features and the target to be predicted, which is travel time. The goal is to be able to predict travel time between any location pair, given a number of locations selected from the datasets. The performance of both models was then tested using accuracy metrics: RMSLE, MAE and a graph of residual distribution to visualise the distribution of the residual count over both datasets. This step is handled in section 4.2 with more details.

**Step 3: Conducting Experiments on three Meta-heuristics**. The predicted travel time between location pairs from the previous step is then saved in a cost matrix. The objective function solves the TSP formulated as finding the route with the shortest travel time by passing through all locations and returning to the start location in the end. Several experiments were performed to find the optimal parameter value combination for each meta-heuristic that gives the best results in terms of the shortest trip duration. Then, the cost matrix and the objective function are used as inputs for the three algorithms, alongside assigning optimal parameter values specific to each meta-heuristic. Each experiment gives the cost of the objective function; in other words, the travel time, the order of locations to visit and their respective addresses, the computation time and a visualised convergence behaviour. The three algorithms were tested on both datasets with problem instances of size: 25, 50, 75 and 100 each with different parameter value combinations.

**Step 4: Experimental Analysis and Visualisation**. This step covers a statistical analysis of the results obtained from the three meta-heuristics. The experiments were carried out 5 times for each problem instance and an average for the results and for the run time was calculated. Finally, a visualised statistical comparative analysis of the results was was performed in chapter 6.

## 3.2 LIMITATIONS AND USE-CASES

We could not find sufficient and reliable data from logistics companies regarding delivery plans. Hence, real taxi demand data from the NYC and Porto datasets were chosen since they are closely related to our problem statement in testing the algorithms. The primary goal is to address finding a solution to the simplest form of TSP, which can be further extended by adding problem-specific constraints that reflect real-world situations. Therefore, drop-off and pickup locations, whether for passengers or parcels, are all treated equally as graph nodes. They are further used alongside temporal data such as pickup time and date to forecast the duration of a trip in minutes which correlate to edges with weights between the nodes. The goal is to use this predicted data to construct the Hamiltonian cycle with the shortest travel time using each of the the three meta-heuristics where a vehicle starts from a node, and visits all nodes once and returns to the start node by the end of the trip. The following examples are real-life use-cases that extend the basic VRP with problem-specific constraints:

1. **Simple Ride-sharing:** A vehicle driver receives multiple ride requests beforehand and the algorithms plan a trip route for pickups and drop-offs with the shortest trip time. Defining a time limit can be added later as an extension to the planning process as described in [Ren+20].
2. **Last-mile delivery:** A transportation vehicle collects parcels from a depot and distributes these parcels to customers. A maximum vehicle capacity can also be added here as a constraint. A survey on last-mile delivery solution approaches is presented by [BG05].
3. **Integrating parcel delivery with passenger transportation:** This scenario incorporates parcel delivery into ride-sharing. Specifying vehicle capacity and time windows can be added as constraints to the algorithm to match the real-world scenario. This VRP use-case was adopted by [Li+14].

# 4

# DATA ANALYSIS

NYC Taxi dataset is frequently used in literature, especially for VRP such as in [San+14] and [Ota+16], and Porto City Taxi dataset was used in [LZL21] to investigate a routing-related problem. Both datasets constitute real taxi demand data and provide all the necessary attributes required to make predictions between location pairs. Therefore, these datasets were found to fit the static route planning case where passenger transportation streams and parcel delivery requests are known beforehand and integrated together in a single network as proposed by [Li+14]. However, considering the time needed to pick up or drop off parcels is not considered and falls out of the scope of this work as the primary goal is to address finding a solution to the simplest form of TSP, which can be later further extended by adding problem-specific constraints that reflect real-world situations.

The first step in the pipeline is to pre-process and perform feature engineering on the available datasets: the NYC Green Taxi Trip Records [2] and Porto Taxi Dataset [3]. This chapter describes the steps taken to prepare the dataset before the training the data to predict the trip duration between coordinate pairs. The first step is to remove the outliers which are identified according to statistical analysis, data dictionaries and information obtained through brief research made over both cities; New York and Porto to reduce prediction error. The second step provides an explanatory analysis of the dataset and the relationship between trip duration and other features within the dataset.

## 4.1 NYC TAXI AND PORTO TAXI DATASETS

### 4.1.1 *Overview*

We construct a real-world routing problem using historical data from Green Cabs in NYC Taxi and Limousine Commission and Porto City Taxi. NYC Taxi dataset was obtained from NYC Government and contains 1,404,726 cab rides taken throughout the month of June in 2016 with 21 attributes, while the original Porto Taxi dataset contains records for a whole year from 01/07/2013 to 30/06/2014 with 9 attributes. However, we took only data between 01/07/2013 and 31/12/2013 which include 858,691 cab rides.

2 https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

3 https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data?select=train.csv.zip

The following are the categorised attributes associated with each dataset:

1. NYC Taxi Dataset Attributes
   - ▷ Administrative: *Vendor ID, Store and fwd flag, Rate Code ID, Trip type*
   - ▷ Temporal: *Pickup datetime, Dropoff datetime*
   - ▷ Geospatial: *Pickup longitude, Pickup latitude, Dropoff longitude, Dropoff latitude, Trip distance*
   - ▷ Payment: *Fare amount, Extra, MTA tax, Tip amount, Tolls amount, E-hail fee, Improvement surcharge, Total amount, Payment type*
2. Porto Taxi Dataset Attributes
   - ▷ Administrative: *Trip ID, Call type, Origin call, Origin stand, Taxi ID*
   - ▷ Temporal: *Timestamp*
   - ▷ Geospatial: *Timestamp, Polyline*

### 4.1.2 *Data pre-processing and feature engineering*

For NYC dataset, we changed all the column names to lower case to start pre-processing conveniently then we added a few extra columns and changed the type of some to facilitate the filtration process. We processed the pickup and dropoff date-time by extracting the date, month, day of week, day of month, hour and minute, and added these new features to new columns. From those newly added features, we calculated the trip duration in minutes and in seconds and added them to new columns as well. However, Porto dataset required more processing as trip data was provided through a polyline which is a list of GPS coordinates recorded at a 15 seconds interval throughout the whole trip. In the last step, Manhattan trip distance and average trip distance were calculated and added to separate columns. We processed the polyline and extracted data needed in the following steps:

1. Extracted the first element in the polyline which corresponds to pickup date time and pickup coordinates
2. Added pickup coordinates in a separate column
3. Converted pickup datetime to pickup date, day of week, hour and minute and added each extracted feature to a separate column
4. Calculated the sum of coordinates in the polyline list.
5. Multiplied the sum of the GPS coordinated in the polyline list to 15 then divided by 60 seconds to get the trip duration in seconds and in minutes.
6. Added trip duration column and a drop off time column which is calculated by adding pickup time to trip time.
7. Processed the the newly added dropoff time column to get dropoff date, day of week, hour and minute and added each extracted feature to a separate column
8. Calculated trip distance in km by finding the sum off distances between each consecutive coordinates in the polyline list.

### 4.1.3 *Data filtering*

**Categorical and administrative data**: First, we started by filtering out rows with missing data and rows with anomalous data entries. According to the data dictionary [4] for NYC dataset, the passenger count within a cab cannot exceed 6 passengers, extra costs can only have 4 different values, while rate code IDs only 6 and lastly the MTA Tax can have only 2 different values. Moreover, we filtered out entries with longitudes and latitudes lying outside New York city's coverage. According to [Loy19] the coordinate ranges of NYC are: latitude [40.63: 40.85] and longitude: [ -74.03 : -73.75]. This step excluded a total of 94564 entries.

According to the data dictionary [5] of Porto dataset, the call type can have only 3 different character values. Origin call has a unique int identifier or NULL value in case the call type was of type "A". origin stand has a unique trip identifier if the call type is "B", otherwise NULL. Moreover, we filtered out entries with longitudes and latitudes lying outside Porto City coverage. Coordinate ranges of Porto City are approximately[6]: latitude [41.0 : 41.3], longitude: [-8.3 : -8.7].

**Trip duration**:

1. NYC Dataset

   As shown in Figure 4.1, there are a lot of trip durations as low as 0 minutes and as high as 1440 minutes. However, by calculating the 99th percentile we get a maximum value of 78.2 minutes. Trips of 0 minutes duration could have been cancelled by the passenger therefore we filtered them out. Furthermore, we excluded trips with a duration of more than 120 minutes. In the last step, we removed trip duration observations that are more than five standard deviations away from the mean duration time. A total of 35208 records were deleted and after eliminating these outliers, we got an average trip duration of 13.1 minutes, a median of 10.4 minutes and 95% of the trip durations fell between 2.42 and 39.4 minutes. These two diagrams are displayed over a logarithmic scale to show the frequency distribution of trip duration before and after the processing. The first histogram shows the anion-uniform distribution and anomalous values reaching 1400 minutes along the x-axis. In histogram B, the distribution is shifted horizontally along the x-axis with a notably different spread of values.

2. Porto Dataset

   There are a lot of trip durations as low as 0 minutes and as high as 958 minutes. However, by calculating the 99th percentile we get a maximum value of 47.5 minutes. Trips of 0 minutes duration could have been be cancelled by the passenger therefore we filtered them out. Furthermore, we excluded trips with durations more than 80 minutes and less than 3 minutes. The two diagrams in Figure 4.2 show the frequency distribution of trip duration before and after the processing using a logarithmic scale. The first histogram shows

4 https://www1.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_green.pdf
5 https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data?select=train.csv.zip
6 https://www.countrycoordinate.com/city-porto-portugal/

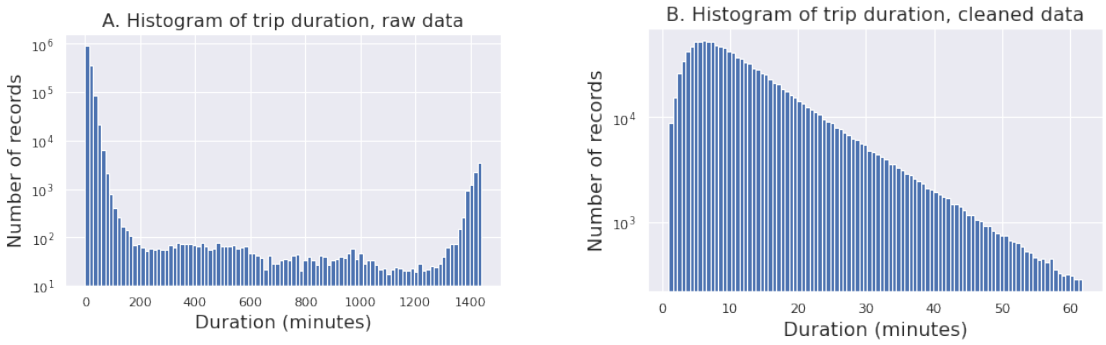



Figure 4.1: Pre-processing Trip Duration - NYC Dataset

an extremely right-skewed distribution closely clustered with a lot of anomalies almost reaching 1000 minutes on the x-axis, while the majority of the trips are under 50 minutes. In Histogram B of the cleaned data, there's better distribution of frequencies with much less clustering along the x-axis with a maximum of 80 minutes. After eliminating these outliers, we got an average trip duration of 12.1 minutes, a median of 10.2 min and 95% of the trips fell between 3.75 and 30.8 minutes. A total of 22724 trips were excluded in this step.

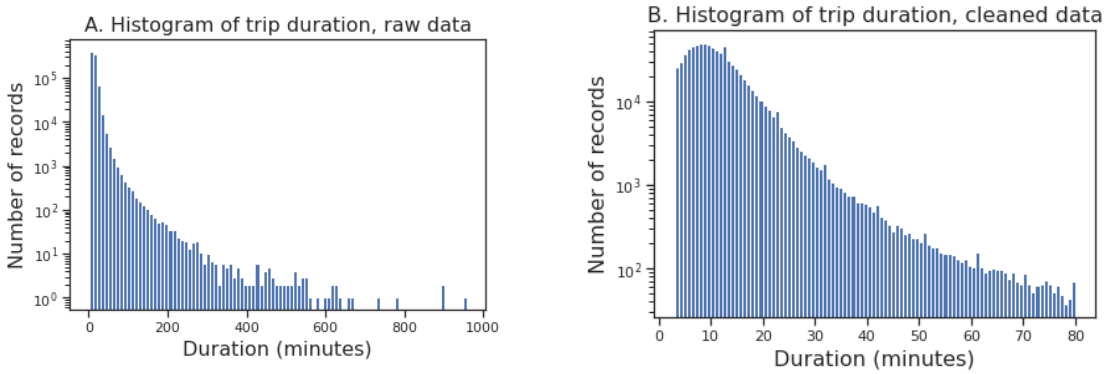



Figure 4.2: Pre-processing Trip Duration - Porto Dataset

**Trip distance**:

1. NYC Dataset

   We removed trips with distances less than 1 mile and more than 30 miles after getting the 99.9th quantile which are apparent outliers. After eliminating these outliers, we got an average trip distance of 3.3 miles, a median of 2.4 miles and 95% of the trips fell between 1.44 and 19.9 miles. These two diagrams show the frequency distribution of trip distance before and after the processing using a logarithmic scale. As illustrated in Figure 4.3, the first histogram shows an extremely right-skewed distribution clustered closely around a few peaks. There are a lot of anomalies lying between 60 miles and 280 miles as displayed in histogram A. Histogram B, the distribution is horizontally shifted along the x-

axis with a notably better spread of values. This step resulted in reducing the trip records by eliminating 247283 trips.

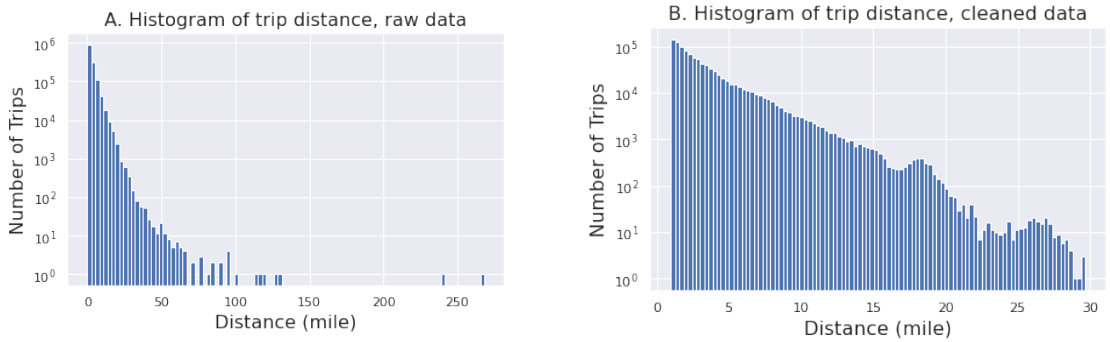



Figure 4.3: Pre-processing Trip Distance - NYC Dataset

2. Porto Dataset

We removed trips with distances less than 1 kilometre and more than 30 kilometres after getting the 99.9th quantile which was an apparent outlier with the value of 43 km. After eliminating these outliers, we got an average trip duration of 6.3 minutes, a median of 4.8 minutes and 95% of the trips fell between 1.44 and 19.9 minutes as shown in Figure 4.4. These two diagrams show the frequency distribution of trip duration before and after the processing using a logarithmic scale. The first histogram shows an extremely right-skewed distribution clustered closely around a few peaks. There are a lot of anomalies lying between 50 km and 1350 km as displayed in histogram A. In histogram B, the distribution is horizontally shifted along the x-axis with a notably different spread of values. A total of 11887 trips were removed.

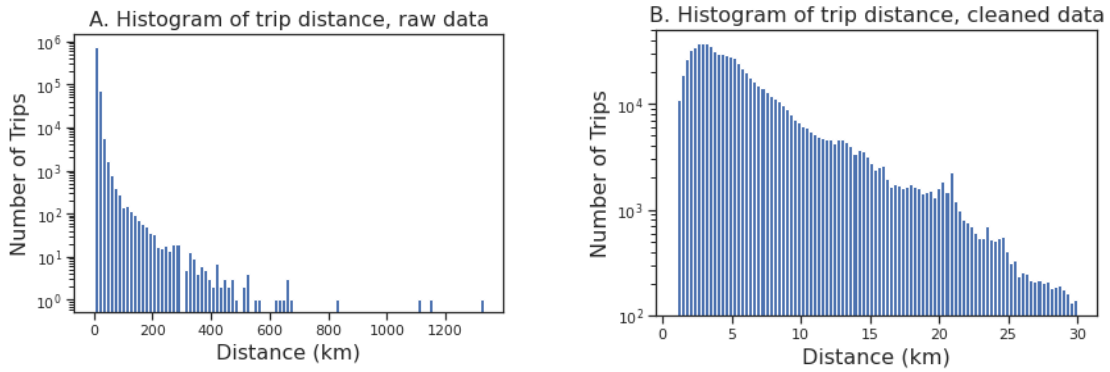



Figure 4.4: Pre-processing Trip Distance - Porto Dataset

**Average trip speed**: We added a new column "trip speed" which corresponds to the average speed of the whole trip. This helped with identifying outliers having extremely low or extremely high trip speeds.

1. NYC Dataset

   The maximum speed limit within NYC is 65 mph [7] so we started by eliminating rows with average trip speeds lower than 5 mph and higher than 55 mph. In the last step, we removed average trip speed observations that are more than five standard deviations away from the mean average speed. We ended up with an average trip speed = 15.3 mph, median = 12.6 mph and 95% of the values fell between 4.85 mph and 41.3 mph. A total of 10495 trips were excluded in this step.

2. Porto Dataset

   First, we added a new column trip speed which corresponds to the average speed of the trip. The maximum speed limit within Porto [8] is 50 km/h in urban areas, 80 km/h on secondary roads and 120 km/h on highways so we will started by eliminating rows with average trip speeds lower than 5 km/h and higher than 100 km/h. In the last step, we removed trip speed observations that are more than four standard deviations away from the mean average speed. After applying these filters, we obtained an average speed of 31.7, median speed 28.3 km/h and 95% of average trip speed values fell between 11.77 and 70.1 km/h. A total of 4010 entries were filtered out.

   A boxplot showing the statistical distribution of the average trip speed values before and after processing the data is displayed in Figure 4.5 and Figure 4.6.

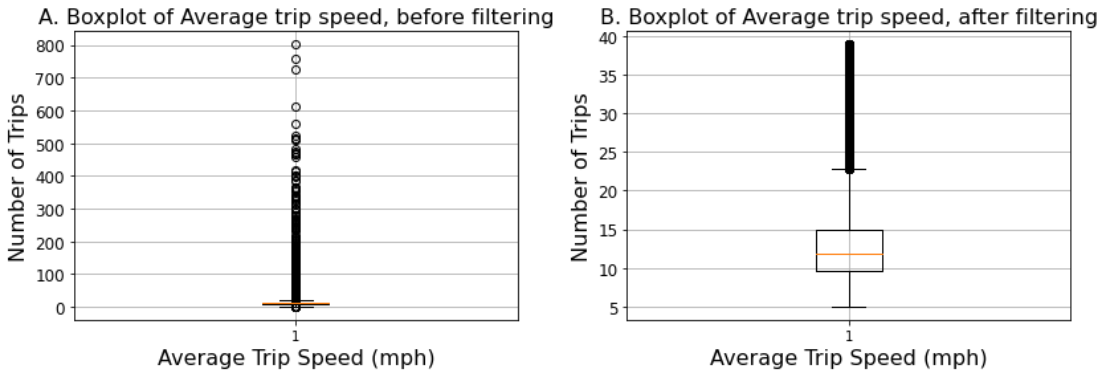



Figure 4.5: Pre-processing Average Trip Speed - NYC Dataset

### 4.1.4 *Explanatory Analysis*

After pre-processing and filtering the dataset in the previous sections, we ended up with 974573 entries and 41 columns for NYC dataset and 791921 entries and 21 columns for Porto dataset. Since trip duration is our target value, we had to investigate

7 https://www.newyorkcarlaws.com/speed-limit/
8 https://www.autoeurope.com/travel-guides/portugal/driving-in-porto//

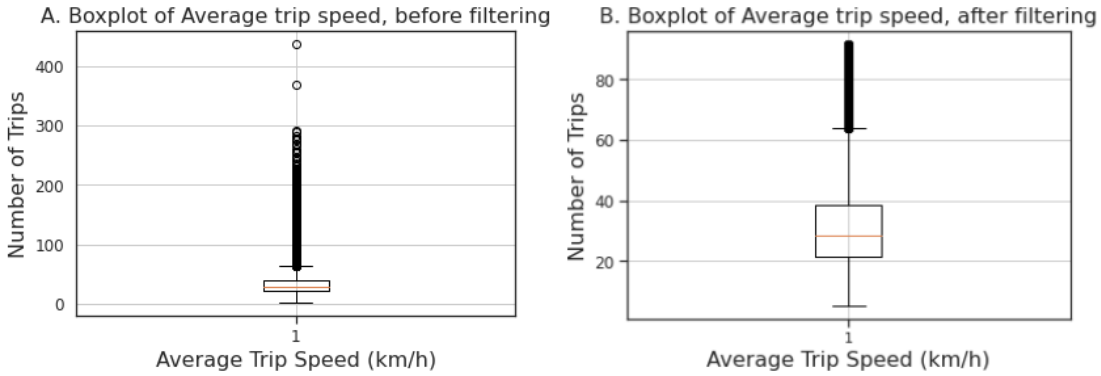



Figure 4.6: Pre-processing Average Trip Speed - Porto Dataset

the degree of influence of the other features on this value. This helped us to decide later which features are essential to train our model to predict the trip duration.

First, in Figure 4.7 and Figure 4.8, we illustrate the trip frequencies distributed over weekdays and hours with both datasets. We observe with both datasets that Saturdays and Sundays have the highest ride counts due to weekend, and with NYC the busiest hours are between 17:00 and 19:00 as this could be the rush hour as people return from their work, while in Porto dataset the busiest hours are between 18:00 and 20:00.

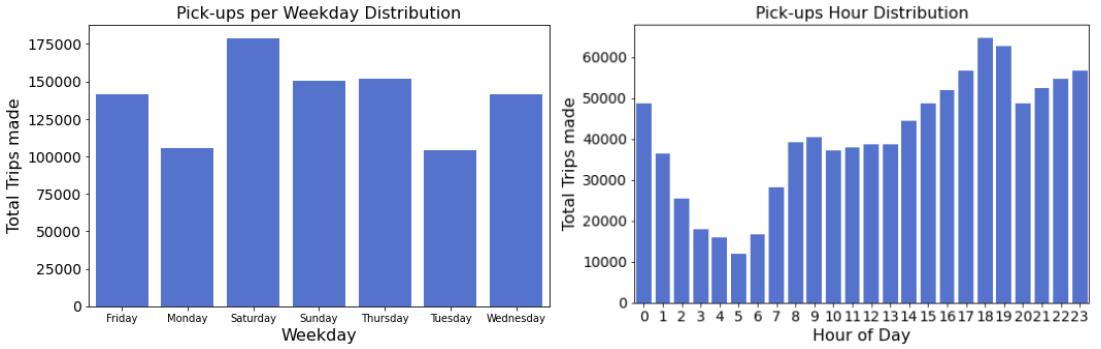



Figure 4.7: Total pickups on each weekday and hour - NYC Dataset

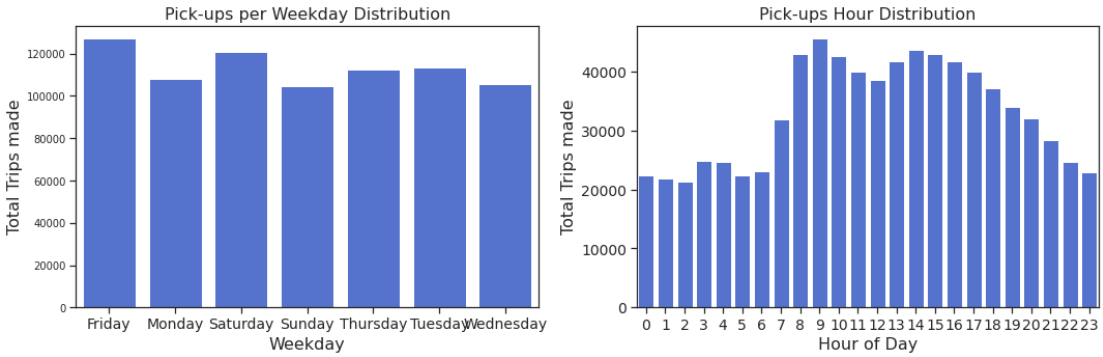



Figure 4.8: Total pickups on each weekday and hour - Porto Dataset

Moreover, we observe a significant variation in the values of trip duration with different trip pickup hours and pickup weekday with both datasets as shown in Figure 4.9 and Figure 4.10. We can therefore draw the conclusion that the timing of the ride has a huge impact in predicting the trip duration, hence pickup hours and weekdays are critical variables in determining the duration of a trip.

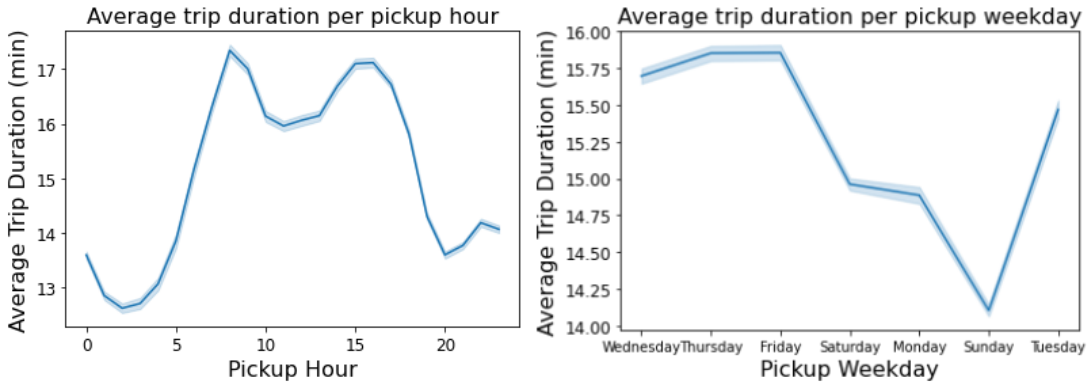



Figure 4.9: Average Trip Duration at different Pickup Hours and Weekdays - NYC Dataset

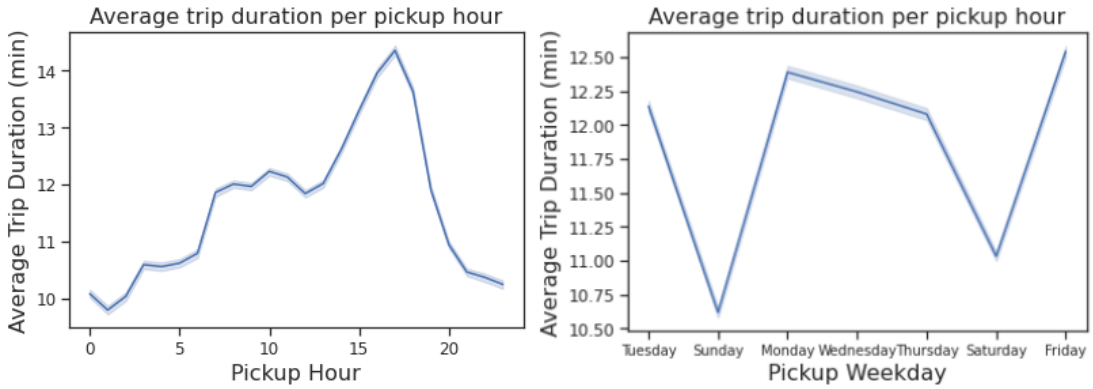



Figure 4.10: Average Trip Duration at different Pickup Hours and Weekdays - Porto Dataset

## 4.2 TRAVEL TIME FORECAST WITH XGBOOST

We concluded from the previous sections that trip durations in both, NYC Green Taxi dataset and Porto City Taxi dataset were heavily influenced by the trip distance, as well as pickup time and pickup day. For the next step in our pipeline, we developed a machine learning model using XGBoost to forecast the trip duration. We used the features mentioned, beside pickup and drop-off coordinates to train our model to predict the trip duration when given a pair of locations. The XGBoost model is saved as a pickle file to be used in the final step in the pipeline. The exact input features are:

**NYC Green Taxi:** pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, trip_distance, pickup_weekday, pickup_hour, pickup_minute.

**Porto City Taxi:** pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, trip_distance, pickup_month, pickup_weekday, pickup_hour, pickup_minute.

To train our model we used 80% of each dataset for training and 20% for testing. Moreover, we trained the model for 500 rounds with the following parameter setting to predict trip duration:

The following parameter setting of XGBoost was used to train the model:

1. *booster = gbtree*: defined the the type of model to be used. Either gbtree for tree-based models or gblinear for linear models
2. *min_child_weight = 1*: controls over-fitting by assigning a minimum sum of weight in all observations in a child
3. *learning_rate = 0.1*: regulises boosting process by shrinking feature weight in each step to prevent overfitting
4. *max_depth = 14*: maximum tree depth to prevent overfitting
5. *subsample = 0.9*: defines fraction of observations to be randomly sampled prior to growing trees. This is to prevent overfitting
6. *n_estimators = 500*: number of decision trees generated for boosting
7. *n_job = -1*: number of parallel threads to the algorithm
8. *silent = 1*: whether to print running messages during the process or not
9. *feval = rmsle*: model evaluation function
10. *colsample_bytree = 0.7* the fraction of randomly selected features that will be used to train each tree

**Model Comparison**:

| | **NYC dataset** | **Porto dataset** |
|---|---|---|
| **RMSLE** | 0.0514 | 0.0633 |
| **MAE** | 2.340 | 2.203 |
| **Max. Error** | 43 | 60 |

Table 4.1: Model Evaluation

For the NYC dataset, the residual frequency is normally distributed around 0 with 0 residuals peak at around 38000 counts. On the other hand, the residual frequency in

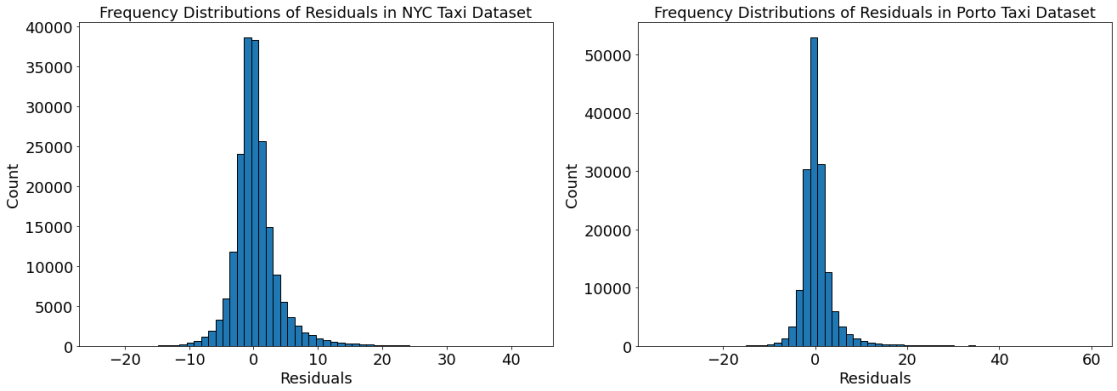



Figure 4.11: Residual Count

Porto dataset is also normally distributed around 0 with a peak of around 50000 counts at a residual value of 0. From the results of the evaluation metrics and the frequency distribution diagram it is observed that the predicted values are slightly more accurate with NYC dataset than with Porto dataset.

# 5

# META-HEURISTICS IMPLEMENTATION FOR OPTIMAL ROUTING

In this chapter, we cover the details of implementing the three meta-heuristics described in section 2.3. We put special attention on algorithm-specific input parameters and the employed convergence approaches. All three meta-heuristics were implemented using scikit-opt library. Scikit-opt is an open source library that provides python modules for several meta-heuristics such as Genetic Algorithm, Differential Evolution, Particle Swarm Optimization, Immune Algorithm and Artificial Fish swarm algorithm, alongside our three meta-heuristics; GA, ACO and SA. The main reasons behind proceeding with this library is that it offers sufficiently good documentation and TSP - specific implementations for GA and SA, and it was relatively easy to integrate the objective function which saves the predicted results in a cost matrix as well. We tested other libraries but unlike scikit-opt, they were found to be difficult to integrate with our objective function and do not exclusively support TSP [9] or lack sufficient documentation and flexibility[10] in testing different parameter values.

## 5.1 GENETIC ALGORITHM

As we mentioned in the previous section, scikit-opt library provides a GA implementation for TSP-related problems. It differs from the general GA implementation mentioned in subsection 2.3.1 by having a distance matrix, which in our case corresponds to a matrix with trip durations between each pair of consecutive locations as well as different crossover and mutation operators, which are described below.

**Parameters**: The algorithm has the following parameters:

- ▷ *func:* objective function to calculate the path with minimum duration.
- ▷ *n_dim:* corresponds to the number of locations that we want to include in the route plan. It must correspond to the number of locations set in the objective function too.
- ▷ *size_pop:* the size of the population corresponds to the number of individuals within the search space. If the population number is low, then there is a higher probability to reach a local optimum. However, if the size is large, the computational load gets higher.
- ▷ *max_iter:* is the number of generations before terminating the process.
- ▷ *prob_mut:* is the mutation factor which is a number between 0 and 1.

Finding the right parameter values determines whether the GA finds an efficient near-optimal solution or not. After setting the parameters, we pass them to the GA function

9 https://pypi.org/project/ACO-Pants/
10 https://github.com/rameziophobia/Travelling$_$Salesman$_$Optimization

and call the main function.
An example for the implemented algorithm[11] is illustrated in Listing 5.1 .

Listing 5.1: GA main function implementation

```
num_points = 49
population = 30
generations = 800
mutation_factor = 1

start_time = time.time()
ga_tsp = GA_TSP(func=total_cost_from_path, n_dim=num_points, size_pop=population,
    max_iter=generations, prob_mut=mutation_factor)
best_path, best_time_cost = ga_tsp.run()

print('Final cost: {} minutes, path: {}, for {} generations' .format(
    best_time_cost, best_path, str(ga_tsp.max_iter+1)))
print("Execution time: % seconds " % (time.time() - start_time))
```

**Evolution Strategies**:

*Ranking*: Since the main problem is finding a tour with the shortest travel time. Individuals with the lowest fitness score have the highest ranking.

*Selection*: This library uses the tournament selection technique [Mil+95]. It's a widely used strategy that involves randomly selecting a number of individuals from a population then running several tournaments among those chosen candidates and the individual with the highest fitness score will be forwarded to the mating pool. The selection pressure applied by this strategy influences the convergence rate of the algorithm as it is a probabilistic measure of the candidate's likelihood of participation in the tournament. This is defined by the tournament size parameter, which is equal to 3 in our case. Meaning that 3 random individuals from the population will be chosen, then their fitness value will be evaluated and the highest among them is selected.

*Crossover*: The library adopts the partially mapped crossover operator which was first proposed by Goldberg and Lingle in [Gol88]. Two random cut points between a subtour on the chromosomes of each parent are chosen, then the portion between the cut points on the first parent is passed to the offspring and the remaining alleles that have no duplicate values from the second parent are mapped to the offspring. If a conflict takes place, we iterate over the second parent's chromosome and select the next allele which was not copied to the offspring. Figure 5.1 illustrates the steps followed by the partially mapped crossover operator.

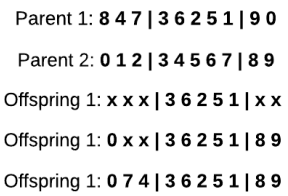



Figure 5.1: PMX Operator.

*Mutation*: First, a random number is generated and if this number is less than the predefined mutation factor then the list of cities in the chromosome is reversed. The reversing process is analogous to the 2-Opt algorithm explained in chapter two where two edges on the chromosome are randomly selected and reconnected so they cross

11 https://github.com/khanhnamle1994/trip-optimizer

over each other.

Listing 5.2 summarizes the GA process:

Listing 5.2: Genetic Algorithm Pseudocode

```
t = 0
generate initial population
evaluate initial population
While{t < max. iteration}
     selection
     crossover
     mutation
     evaluation
     t = t+1
terminate
return best individual in population
```

## 5.2 ANT COLONY OPTIMIZATION

ACO is already a graph-based optimisation approach, therefore it can be easily employed to solve the TSP without any further adaptations. Hence, there was no special version for the TSP defined in the scikit-opt library.

**Parameters**: To run the ACO, we have to pass the following input parameters to the main function:

- ▷ *func:* objective function to calculate the path with minimum duration.
- ▷ *n_dim:* corresponds to the number of locations that we want to include in the route plan. It must correspond to the number of locations set in the objective function too.
- ▷ *size_pop:* the size of the population corresponds to the number of ants within the search space. If the population number is low, then there is a higher probability to reach a local optimum. However, if the size is large, the computational load gets higher.
- ▷ *max_iter:* is the number of iterations before terminating the process.
- ▷ *alpha:* a parameter to control influence of pheromone level on the probability to select a specific path.
- ▷ *beta:* a parameter that states the degree of influence of the desirability state transition on the probability to select a specific path.
- ▷ *rho:* pheromone evaporation rate.
- ▷ *distance_matrix:* corresponds to the cost_matrix from the objective function which records all the predicted durations between each location pairs tested with ACO .

**Mode of Operation**: As a result of the main function execution, a pheromone matrix and a path matrix to record the path taken by each ant in each generation are constructed. In each iteration, the path taken is recorded in the path matrix and values within the pheromone matrix are updated. Then, the best total travel time within each generation is calculated and updated in the following generations, and the best

shortest travel time value is updated.
Listing 5.3 summarizes the ACO process:

Listing 5.3: Ant Colony Optimisation Algorithm Pseudocode

```
t = 0
iteration i = 0
initialise pheromone trail values
place ants at random cities
while i < max. iteration
     for each ant
        Repeat
            Calculate probability p
            Select next unvisited city to visit next based on state transition
                probability
            Update local pheromone value
        Until
            all nodes are visited or a tour is constructed
    update global pheromone value
return best solution
```

## 5.3 SIMULATED ANNEALING

The library offers SA for TSP which differs from the conventional SA in utilising a logarithmic cooling schedule and in using different methods to explore neighbouring solution. These methods are explained in details throughout this section.

**Parameters**: To run the simulated annealing, we have to pass the following input parameters to the main function:

- ▷ *func:* objective function to calculate the path with minimum duration.
- ▷ *T_max:* Maximum (Initial) Temperature.
- ▷ *T_min:* Minimum (End) Temperature.
- ▷ *L:* Long of chain or number of iterations at each temperature.
- ▷ *n_dim:* number of locations to be visited.
- ▷ *max_stay_counter:* cool down time. Number of iterations before termination when no change is observed in the solution value.

**Mode of Operation**:
*Cooling Schedule*: As we mentioned before, an effective cooling schedule is the major determinant in reducing the execution time before finding the global minimum and the cooling rate specifically, plays a crucial role in the algorithm's ability to find the global minimum. A logarithmic reduction scheme is adopted here in the implementation of the cooling schedule. This cooling schedule behaves as: $T(t) \sim 1/\log t$, and according to some studies, this method guarantees converging to the global minimum but in the limit of infinite time. To counteract this issue, the implemented algorithm uses the parameter max_stay_counter to terminate after a specific number of iterations when

no further changes have been observed in the solution value. It restricts the algorithm to simulate the active portion of the decay and not the frozen periods.

*Candidate Generation Strategy*: There are three different simple strategies for the algorithm to generate new neighbouring solutions and only one of those three is randomly selected:

1. Swapping: Selecting two random locations and swapping their positions in the array.
2. Reversing: Selecting two random edges and reconnecting them so that they cross.
3. Transposing: Slicing the original tour into four unequal sections and rearranging them in a different order.

*Acceptance Probabilities*: The new solution is adopted instead of the current solution if one of those two criteria is met:

1. The cost of the new solution is less than the old solution.
2. The cost of the new solution is more than the old solution and this probability is satisfied:
$$e^{(-\Delta k/T_c)} > r$$
where $\Delta k$ denotes the cost difference between old and new solution, $T_c$ stands for the current temperature and $r$ is a randomly generated integer.

Listing 5.4 illustrates the pseudocode for SA.

Listing 5.4: Simulated Annealing Pseudocode

```
generate initial solution S
iteration = 0
stay_counter = 0
T = T_max
for i in range L
    generate candidate S'
    delta_c = S' - S
    generate random number r

if delta_c < 0  or  e^{(- delta_c / T_c)} > r}
    S = S'
if S < best_S
    best_S = S

iteration = iteration + 1
cool_down()

if best_S = best_S[iteration-1]
    stay_counter = stay_counter + 1
else
    stay_counter = 0

if T < T_min then terminate

if stay_counter > max_stay_counter then terminate

return solution
```

# 6 EXPERIMENTAL RESULTS AND DISCUSSIONS

This chapter provides an overview of the system specifications, description of the experimental setup and the analogy adopted to test the three mentioned algorithms. The obtained results from each experiment are presented, analysed, and a comparative analysis of the three optimisation algorithms is performed.

## 6.1 IMPLEMENTATION

### 6.1.1 *System Specifications*

The three meta-heuristics were implemented in Python 3 using scikit-opt library on the browser-based notebook Colab provided by Google Cloud Platform. Colab is an online Jupyter notebook integrated with Google Drive and allows combining executable python code with text in a single document. Colab is free, requires no configuration and provides 13 GB RAM and 2vCPU @ 2.2GHz.

### 6.1.2 *Software Description*

For each dataset, 2 separate notebooks were used; one for preparing the dataset and one for training the model and testing the three meta-heuristics. The notebook for data pre-processing is organised in the general structure with the following blocks:

1. Loading and carrying out statistical analysis of the raw dataset.
2. Exploring the columns, data types, missing and unique data and adjusting the dataset accordingly by feature engineering to prepare the dataset for a more acurate analyss and filteration process.
3. Filter out anomalies based on the data dictionary or a statistical estimation and visualising important features in the dataset before and after pre-procesing.
4. Carrying out a final general exploratory analysis of the pre-processed dataset.

The second notebook starts by loading the pre-processed dataset and has the following logical structure:

1. Loads the pre-processed dataset and installs XGBoost.
2. Trains and saves the XGBoost model using essential features in predicting travel time. It also summons evaluating the trained model using evaluation metrics that were implemented (RMSLE and MAE).
3. Loads the saved model, installs scikit-opt library to test the algorithms and includes the implementation of the objective function which is split into: a function to convert distance between coordinates to Manhattan distance and use it as an input feature for another function to predict time cost between location pairs as well as a function that saves the predicted values in a cost matrix and calculates the total path cost. In addition, a function to select any number of

locations with their indices from the dataset to perform prediction on each pair of the selected location points is implemented.

4. Consists of sub divisions each of which for corresponding to a distinct meta-heuristic which was implemented with the help of scikit-opt library. The number of locations and predicted distance saved in the cost matrix were carried forward to be used as an input for each meta-heuristic. Moreover, a function to calculate time needed to find optimal path in each iteration along with functions to calculate the travel time and the locations in form of indices and geographical addresses were also implemented. Finally, we added functions to plot the convergence behaviour and the route coordinates.

Lastly, a fifth notebook was added to test brute force algorithm on both datasets.

### 6.1.3 *Experimental Setup*

In chapter 4, we discussed training and using a XGBoost model to predict trip durations. This chapter covers the last stage of the pipeline, where we assign numerical values to the input features that are used in a function to predict dependent variable (travel time) between every pair of locations selected from each dataset. The numerical values of the independent parameters in the datasets are presented in Table 6.1. No pickup month feature is assigned to NYC Dataset, since the dataset includes entries during the month of June only, therefore there is no need to add pickup month as an independent variable.

| Parameters | NYC dataset | Porto dataset |
|---|---|---|
| pickup_weekday | 3 | 1 |
| pickup_hour | 15 | 18 |
| pickup_minute | 40 | 34 |
| pickup_month | – | 8 |

Table 6.1: Parameter values

Trip distance cost was also calculated in both datasets using a function that calculates Manhattan distance between consecutive locations from their coordinates and then used the distance as input parameter in the prediction function.

As shown in Listing 6.1, the predicted values were then added to a cost matrix where the predicted distance between each pair of locations is saved and used later in the objective function[12] that calculates the minimum path cost.

Since we investigated four different location counts n: 25, 50, 75 and 100, the first n locations in each dataset were selected. We compared the simulation results of the three meta-heuristic algorithms on the NYC and Porto datasets. In order to carry out the experiments, multiple factors were considered, including problem size and other algorithm-based parameters. We tested each algorithm on different problem sizes corresponding to location counts. The problem sizes are {25, 50, 75, 100} for each set of locations or problem size the parameters of the three algorithms were tested using

12 https://github.com/khanhnamle1994/trip-optimizer/blob/master/Bio-Inspired-Algorithms/genetic_evo_main.py

Listing 6.1: Building a complete cost matrix based on predicted time between points

```
locations = []
points = []

#take the first 50 rows within the dataset as input coordinates
for index, row in df.iloc[:49].iterrows():

        locations.append({
              'index': index,
              'x': row['pickup_longitude'],
              'y': row['pickup_latitude']
          })
        points.append((row['pickup_longitude'], row['pickup_latitude']))

# Build complete cost matrix based on time between points
cost_matrix = []
rank = len(locations)
for i in range(rank):
    row = []
    for j in range(rank):
        row.append(time_cost_between_points(locations[i], locations[j],  3, 15,
            40))
    cost_matrix.append(row)
print(cost_matrix)
```

different values and tuned to obtain the best results possible in order to carry out a fair comparison. After finding the most fit set of parameters for each algorithm, the experiments were then repeated five times and the results were recorded for each set of locations, and the average and standard deviation were calculated. The purpose of collecting many samples is to test the reliability of the obtained results.For each experiment the following data was collected: **(1) Runtime (s)**; **(2) Trip duration (m)**; **(3) Minimum cost** (mean, standard deviation). The following comparison metrics aided in comparing the results after repeating the experiments five times:

1. **Runtime** (s) (minumum, maximum, average, standard deviation)
2. **Trip duration** (m) (minimum, maximum, average, standard deviation)

The simulation results were then recorded based on the previously mentioned criteria adopted to evaluate the experimental outcomes.

## 6.2 GENETIC ALGORITHM

### 6.2.1 *Parameter tuning*

The GA generally uses two fundamental parameters. The first parameter is population size which is concerned with selecting the most fit individuals from the whole population in each generation. This parameter regulates the selection pressure applied on each generation where the balance between exploring the solution space to obtain more diversity and continuing the exploitation of the already detected results to converge towards an optimum is crucial. Mutation factor is the second parameter and

it determines the size of genetic component on the chromosome on which a random change in values takes place enabling a wider solution space exploration by introducing diversity. However, the algorithm written in this library does not implement a mutation factor in this sense. In the library, first, a random number is generated and if this number is less than the predefined mutation factor then the list of cities in the chromosome is reversed in a way analogous to the 2-Opt algorithm. Therefore, experimenting on changing the mutation factor would not play any significant role in the quality of the results.

**Population Size:** Selecting an adequate population size in each generation prevents the premature convergence which is the primary issue with the genetic algorithms. A small population size guarantees a quick, yet premature convergence, while a high value is computationally expensive. Therefore, it is important to find the adequate size which maintains the balance between solution quality and efficiency. The effect of changing the population size on the trip duration was tested on four problem sizes from the set: {25, 50, 75, 100} by running the algorithms with 10 different population sizes ranging from 10 to 140 for each location count. Further, the population size resulting in lowest trip duration was selected for each problem size tested. Figure 6.1 summarize the results obtained with different population sizes on both datasets and the results of the experiments on NYC and Porto datasets are presented in Table A.1 and Table A.2 in the appendix.


Figure 6.1: Population Size vs. Trip Duration

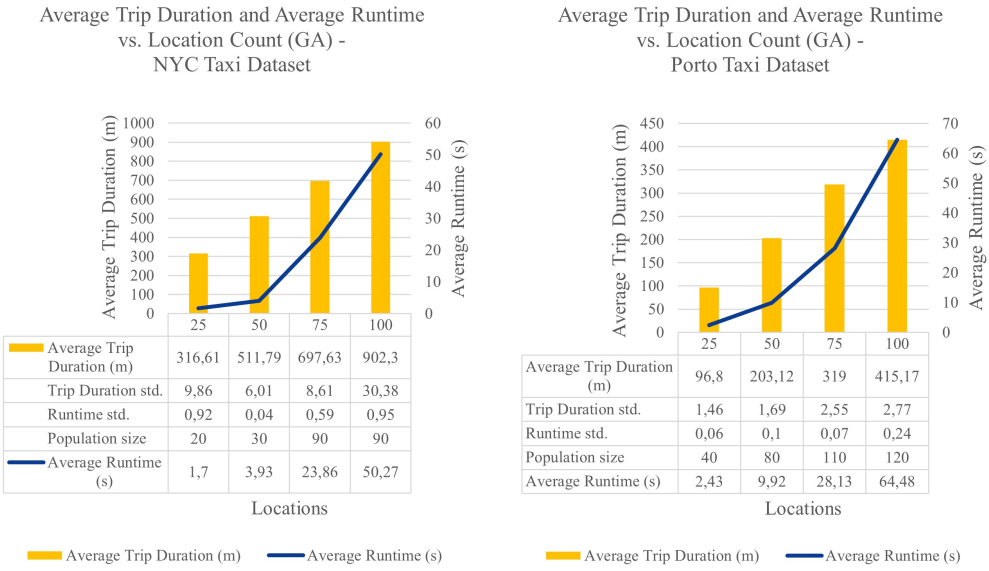
It is trivial that the population size increases with the increase in location count as the selection pressure needs to be adjusted to fit with a larger solution space. Table 6.2 summarises the optimal population sizes found for each problem size:

Table 6.2: Optimal Population Sizes and generations for the GA

| Locations | Population size: NYC | Population size: Porto | Generations |
|---|---|---|---|
| 25 | 20 | 40 | 600 |
| 50 | 30 | 80 | 800 |
| 75 | 90 | 110 | 1200 |
| 100 | 90 | 120 | 2000 |

**Number of generations:** The number of generations determined for each location count is approximated and chosen based on the certainty that the algorithm does not converge beyond this number. Therefore, we settled on the generation number for both datasets provided in Table 6.2.

### 6.2.2 *Algorithm Results*




| | 25 | 50 | 75 | 100 |
|---|---|---|---|---|
| Average Trip Duration (m) | 316,61 | 511,79 | 697,63 | 902,3 |
| Trip Duration std. | 9,86 | 6,01 | 8,61 | 30,38 |
| Runtime std. | 0,92 | 0,04 | 0,59 | 0,95 |
| Population size | 20 | 30 | 90 | 90 |
| Average Runtime (s) | 1,7 | 3,93 | 23,86 | 50,27 |

| | 25 | 50 | 75 | 100 |
|---|---|---|---|---|
| Average Trip Duration (m) | 96,8 | 203,12 | 319 | 415,17 |
| Trip Duration std. | 1,46 | 1,69 | 2,55 | 2,77 |
| Runtime std. | 0,06 | 0,1 | 0,07 | 0,24 |
| Population size | 40 | 80 | 110 | 120 |
| Average Runtime (s) | 2,43 | 9,92 | 28,13 | 64,48 |

Figure 6.2: Trip Duration with GA

Figure 6.2 illustrates the change in the average trip duration and average runtime for each set of locations on both datasets. Despite demonstrating a significantly higher average runtime with Porto dataset than with NYC dataset, especially with large problem sizes starting from 50, the algorithm appears to produce more accurate results with the Porto dataset. The standard deviation values for the trip duration in NYC dataset in comparison to Porto dataset are much higher, indicating a wide range of variance within the solutions each time the algorithm is executed. This, however, implies that the GA algorithm gives more reliable results with the Porto dataset. Nevertheless, the algorithm seems to exhibit a slow exponential behaviour with the runtime as the problem size increases.

| Locations | Population Size | Iterations | $\alpha$ | $\beta$ | $\rho$ |
|---|---|---|---|---|---|
| 25 | 20 | 800 | 1 | 5 | 0.1 |
| 50 | 30 | 800 | 1 | 5 | 0.1 |
| 75 | 40 | 800 | 1 | 5 | 0.1 |
| 100 | 40 | 800 | 1 | 5 | 0.1 |

Table 6.3: Optimal parameter values for the ACO - NYC Taxi Dataset

## 6.3 ANT COLONY OPTIMIZATION

### 6.3.1 *Parameter tuning*

The ACO uses a set of parameters that work together to control the importance of the deposited pheromone amounts in deciding the next location to visit in the tour. Beside the number of ants in each iteration *population_size*, parameters that find the shortest path on a probability basis: pheromone coefficient $\alpha$, heuristic coefficient $\beta$ , and evaporation coefficient $\rho$ cannot be treated as independent variables and cannot be tuned in isolation from each other.
**Number of Iterations:** The number of iterations was 800 throughout the whole experimental process. No optimal results were observed beyond 800 iterations.

**Pheromone, heuristic and evaporation coefficients:** Due to the lack of specific criteria for setting up the parameters in ACO , the algorithm was tested with various parameter combinations for each problem size: $\alpha$ = {0.5, 1, 1.2, 2, 5} and $\beta$ = {0.5, 1, 2, 3, 5} while $\rho$ was held constant at 0.1. Experiments to determine the optimal coefficient values were carried out with location count = 50, population size = 30 and number of iterations = 800. The experimental results on both datasets are presented in Table A.3 and Table A.6 in the appendix.

In the following step, the parameter values that produced the best 4 results with each dataset were selected, which are present in Table A.4 and Table A.7 in the appendix. The algorithm was tested again using these 4 different combined parameter values five times for each combination and the average result was calculated. Parameter values that gave the shortest trip duration with each dataset were then selected. As shown in Figure 6.3 below, these values are $\alpha$ = 1 and $\beta$ = 5 for NYC Taxi dataset, and $\alpha$ = 2 and $\beta$ = 5 for the Porto City Taxi dataset.

**Population size:** Finally, the algorithm was tested using different population sizes ranging from 10 to 100 with 10 intervals on each set of locations to find the optimal number of ants to get the best results. According to the test results in Table A.5 and Table A.8, the following values as illustrated in Table 6.3 and Table 6.4 were found to have the best results and were used throughout the rest of the process:

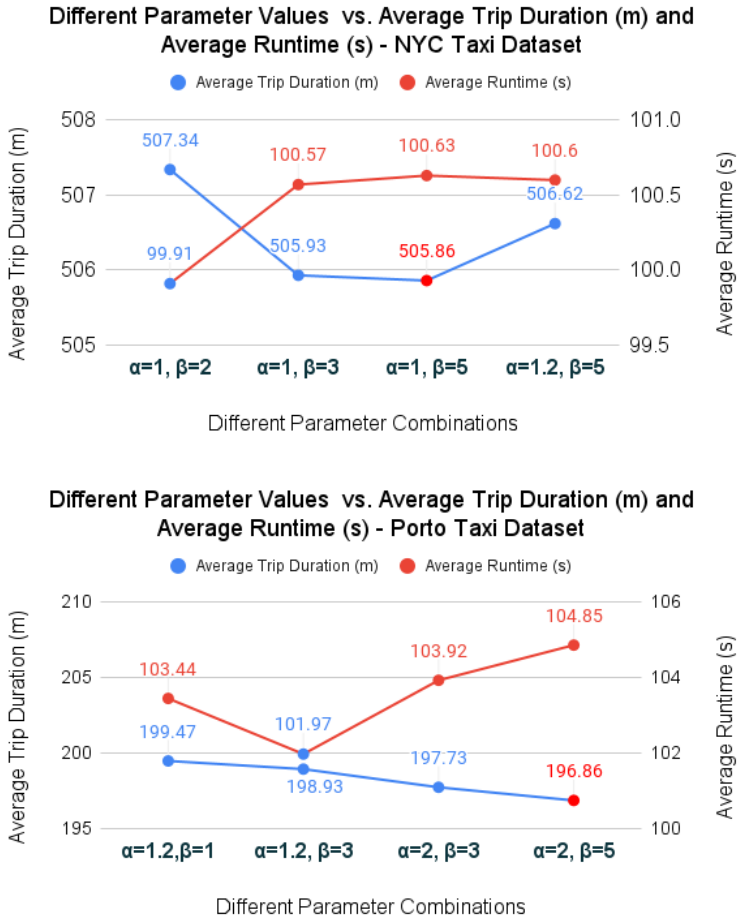



Figure 6.3: Optimal Parameter Value Combinations for both Datasets

### 6.3.2 *Algorithm Results*

As illustrated in Figure 6.4 below, the algorithm shows an overall better performance with Porto Dataset. The trip duration standard deviations are slightly lower in comparison to standard deviations in NYC dataset depicting more accurate results. However, the runtime of the algorithm on NYC Taxi dataset in comparison to Porto dataset maintains a linear behaviour, despite needing more time for execution with 25, 50 and 75 locations. On the other hand, average runtime appears to be slightly exponential with Porto dataset as with 100 locations the difference is much higher. Hence, the algorithm appears to perform more efficiently on NYC dataset when the input size is large, but provides more accurate results with Porto dataset.

## 6.4 SIMULATED ANNEALING

### 6.4.1 *Parameter Tuning*

Like the other meta-heuristics, the ideal parameter values for Simulated Annealing cannot be determined beforehand. The SA parameters are provided as black box

| Locations | Population Size | Iterations | α | β | ρ |
|---|---|---|---|---|---|
| 25 | 20 | 800 | 2 | 5 | 0.1 |
| 50 | 20 | 800 | 2 | 5 | 0.1 |
| 75 | 30 | 800 | 2 | 5 | 0.1 |
| 100 | 50 | 800 | 2 | 5 | 0.1 |

Table 6.4: Optimal parameter values for the ACO - Porto Taxi Dataset

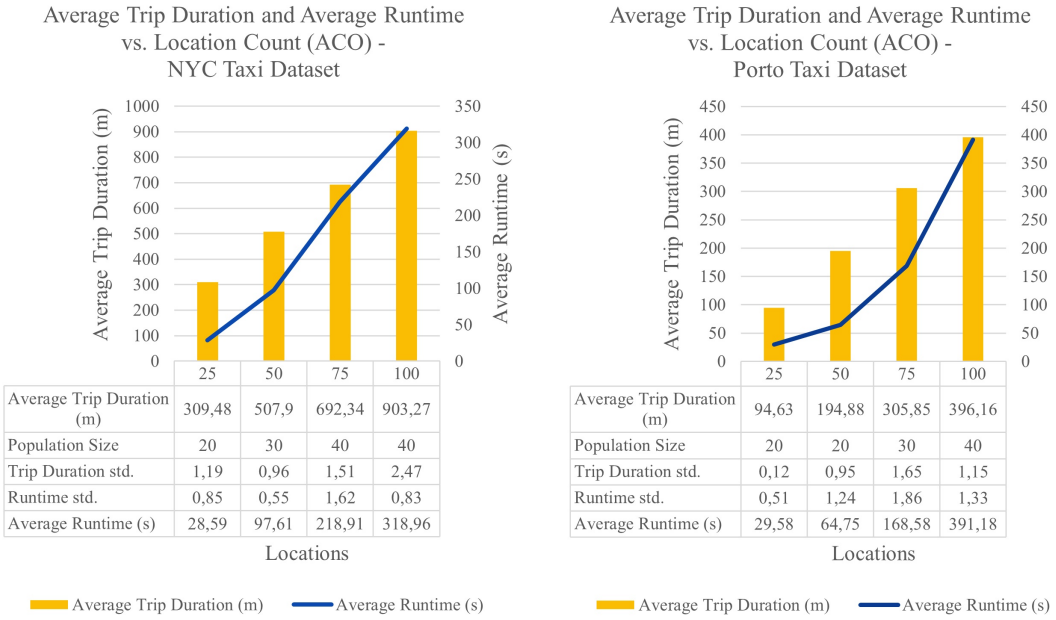



Figure 6.4: ACO Results

functions that have to be empirically adjusted for each problem and hence, with each dataset. We used grid search here for hyperparameter tuning by trying different combination of values.

**Maximum and Minimum Temperature:** The maximum temperature T_max is the initial temperate in the beginning of the annealing process. As mentioned before, higher temperatures allow the algorithm to accept poor results to explore the solution space more as they might lead to overall good results in the end. Very high maximum temperatures however increase the frequency of accepting worse solutions while with very low temperatures the algorithm is more likely to get stuck in local minima. Therefore, the algorithm was tested with different T_max values and the results are presented in Table A.9 and Table A.12 to find the optimal temperature. On both datasets, the optimal initial temperature was 1°C. After determining the maximum temperature, the algorithm was further tested with various Minimum Temperatures T_min which is one of the termination criteria, as the algorithm stops once the cooling schedule reaches the minimum temperature. According to the experimental results in Table A.10 and Table A.13 the optimal T_min value found for NYC dataset is 1.00E-9 and 1.00E-4 for Porto dataset. The following diagrams show the results obtained with different T_max and T_min values for each dataset. The obtained results are displayed in Figure 6.5 below:

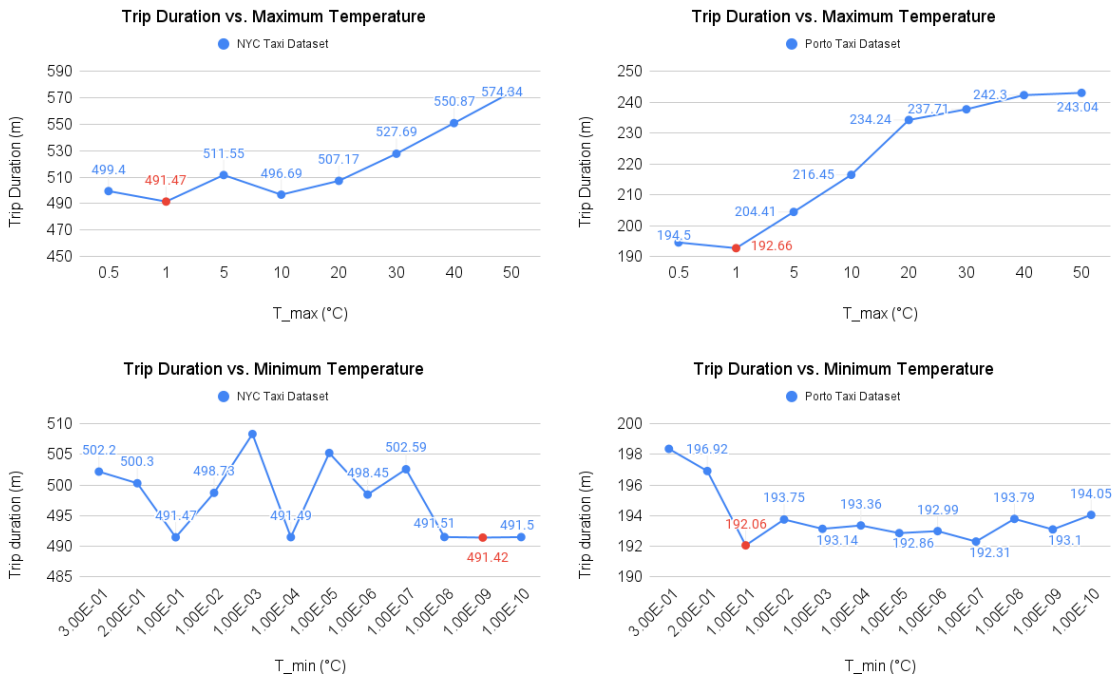



Figure 6.5: Tuning Temperatures

**Number of Iterations and Maximum Counter:** As mentioned in chapter 5, the cooling schedule adopted has a very slow rate and in some cases needs infinite time to reach the minimum temperature and terminate. The library here proposes a solution to counteract this issue. Two extra parameters L and max_stop_counter shape the termination criteria and prevent the algorithm from lasting long, despite finding the optimal or near-optimal solution. The L denotes the long of chain which is an integer multiplied by the number of locations to give a total number of iterations and max_stop_counter is another nested parameter for iterations which forces the algorithm to terminate after a fixed number of iterations, during which no change in the result is observed. Small glitches can happen after the algorithm is believed to converge, these parameters however, when tuned well, ensure that no further improvement in the result is missed out. Both parameters work together and cannot be tuned in isolation. Therefore, the following parameter value combinations were tested on each of the four problem sizes: L = {10, 20, 40} and max_stop_counter = {100, 200, 300, 400, 500} and the experimental results are resented in Table A.11 and Table A.14 in the appendix. Table 6.5 summarises the optimal values found for each set of locations in each dataset.

### 6.4.2 *Algorithm Results*

As illustrated in Figure 6.6 below, the algorithm has a better performance in terms of solution accuracy with Porto dataset as the Trip duration standard deviation values are significantly lower than NYC dataset implying reliable results due to a narrower range of variation in the results. However, the average runtime is significantly higher with Porto dataset with very high standard deviation values indicating impersistence

Table 6.5: Number of Iterations and Maximum Counter

| | NYC Dataset | | Porto Dataset | |
|---|---|---|---|---|
| Locations | L | max_stop_counter | L | max_stop_counter |
| 25 | 10 | 400 | 20 | 300 |
| 50 | 20 | 400 | 10 | 500 |
| 75 | 10 | 500 | 10 | 500 |
| 100 | 10 | 500 | 20 | 400 |

with the execution time especially over large problem sizes starting from 50 locations with Porto dataset.


| | 25 | 50 | 75 | 100 |
|---|---|---|---|---|
| Average Trip Duration (m) | 303,57 | 491,55 | 681,63 | 869,43 |
| Trip Duration std. | 0,21 | 1,82 | 1,6 | 0,96 |
| Runtime std. | 0,68 | 8,72 | 5,07 | 17,7 |
| Average Runtime (s) | 2,41 | 20,75 | 28,5 | 56 |

| | 25 | 50 | 75 | 100 |
|---|---|---|---|---|
| Average Trip Duration (m) | 93,53 | 191,95 | 297,85 | 389,3 |
| Trip Duration std. | 0,06 | 0,41 | 0,54 | 0,4 |
| Runtime std. | 1,9 | 12,69 | 47,15 | 41,15 |
| Average Runtime (s) | 11,59 | 52,77 | 137,84 | 282,24 |

Figure 6.6: SA Results

## 6.5 COMPARISON AND ANALYSIS OF RESULTS

This section covers a quantitative and a qualitative comparison as well as analysis of the results obtained by the three algorithmic approaches; GA, ACO and SA over the both Taxi datasets. First, we consider how accurate can these algorithms determine the average trip duration with different sets of locations. Secondly, compare the execution time of the algorithms. Finally, we draw a conclusion about the efficiency of the mentioned algorithms and provide the recommendations for both datasets.

### 6.5.1 *Benchmark Instance*

Due to the enormous time needed to calculate the optimal solution to the TSP using exact methods, it was difficult to compare the results with baseline optimal solutions.

However, we employed Brute-force algorithm to test the runtime. The algorithm was tested with problem sizes starting from 3 locations and terminated after the $14^{th}$ location as the algorithm took more than 5 hours to calculate the shortest path with 15 locations. As displayed in Figure 6.7 below, for both datasets the algorithm exhibited an exponential growth.

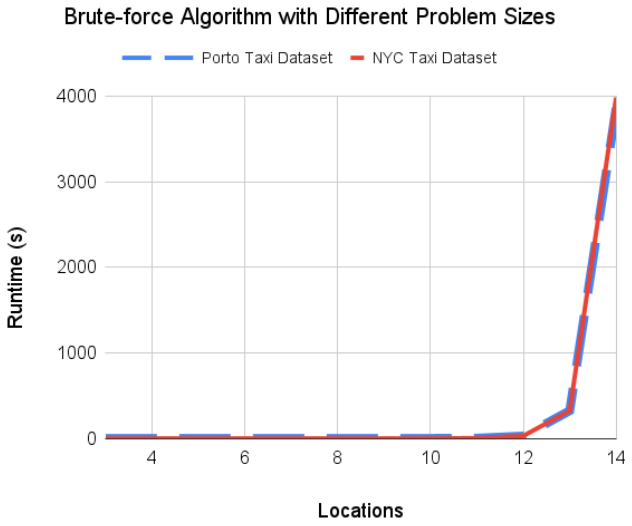



Figure 6.7: Brute Force Algorithm

### 6.5.2 *Algorithms Evaluation*

**Accuracy**: The figure below illustrates the calculated average trip duration with each of the three algorithms on both datasets. SA gave the best results in terms of the shortest trip duration in both datasets as displayed in Figure 6.8. ACO came in the second place and GA came in the third. The differences between the results obtained by SA and the other algorithms are significantly high with large problem instances as observed with 100 locations. On the other hand, the results obtained by ACO were closer to the results obtained by GA than SA in NYC dataset. As the number of locations increased, the difference in results between ACO and the other algorithms almost stayed constant and was not affected. In addition, the difference gap in the results between SA and GA is larger as the location count increases, indicating that the performance of GA is poor with large problem instances.

**Efficiency**: The algorithm efficiency observes and compares the average computation time (Runtime) taken by each of the three algorithms to get the results. The average runtime was calculated from the four different location instances on both datasets in 5 test runs. Based on the results illustrated in Figure 6.11, the average time required by ACO to find the optimal trip duration is higher compared to GA and SA. On the other hand, computation time required by SA to find the optimal solution is notably different in each dataset. In NYC Dataset, SA has a runtime almost similar to GA, while in Porto Dataset, SA starts with runtime almost similar to GA with 25 locations then

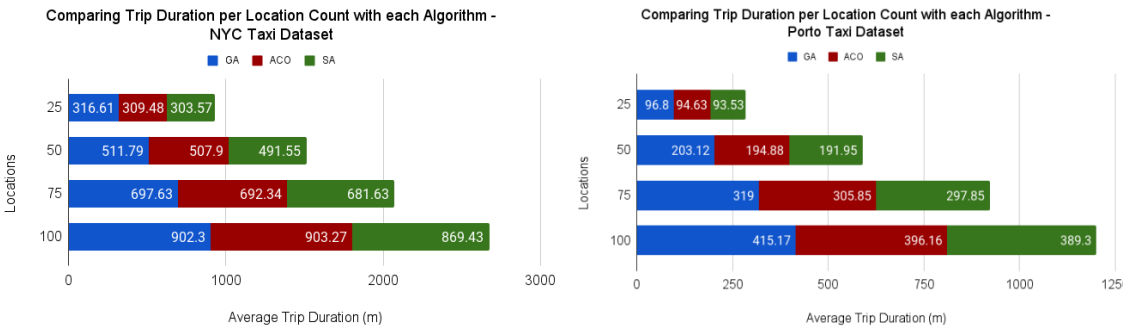



Figure 6.8: Average Trip Duration

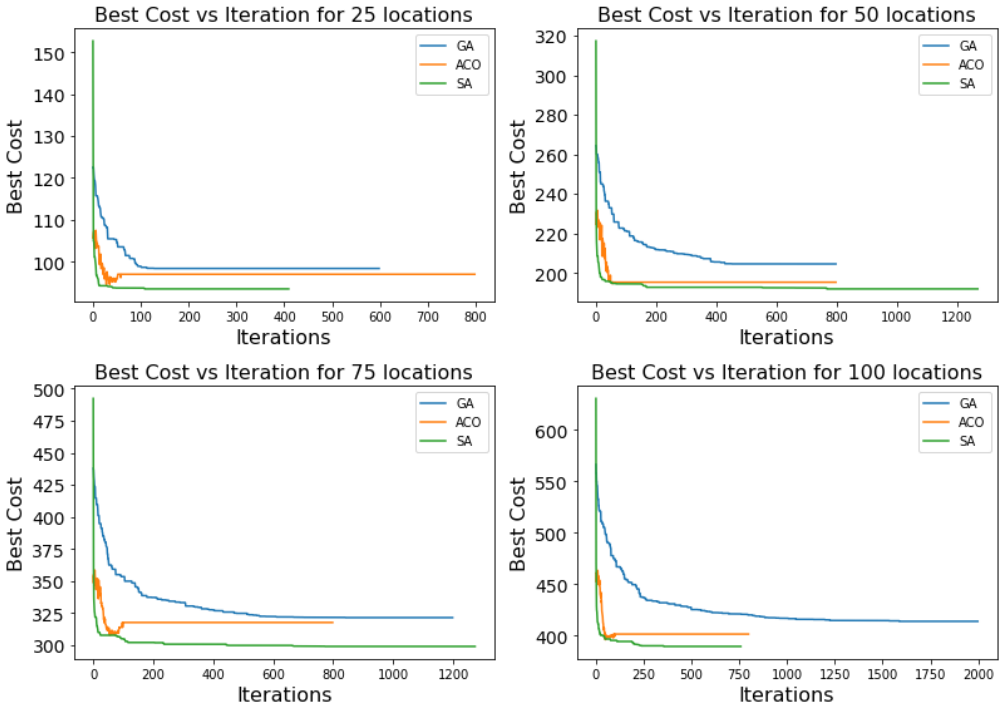



Figure 6.9: Convergence with each location count - Porto Dataset

requires almost same amount of computation time as ACO with 50 and 75 locations, but can take up to 300 seconds, while ACO 400 seconds with 100 locations. Overall, highest efficiency was achieved by GA and ACO had the worst. Moreover, Figure 6.9 and Figure 6.10 illustrate the convergence rates with different problem instances on both datasets. GA exhibited overall the slowest convergence rates, while ACO and SA displayed a relatively faster convergence. However, SA has shown a more stable convergence behaviour than ACO.

**Reliability**: The reliability of the obtained results were measured in accordance to standard deviation values as the experiments were repeated a total of 5 times for each location set on both datasets. Overall, the standard deviation for the trip duration is generally higher for the three algorithms in the NYC dataset where the range lies

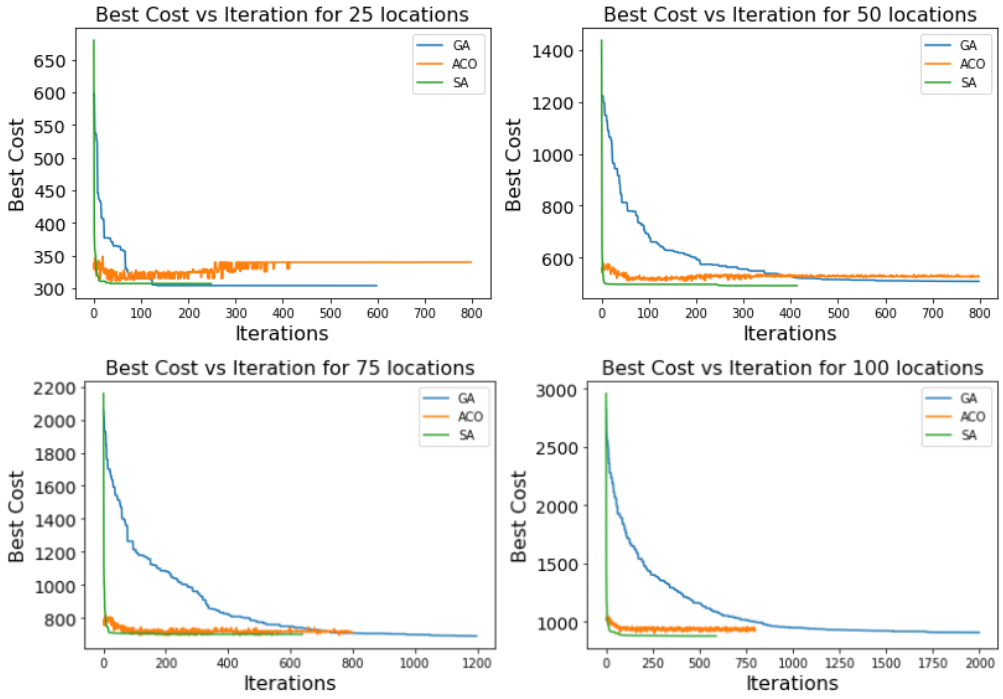



Figure 6.10: Convergence with each location count - NYC Dataset

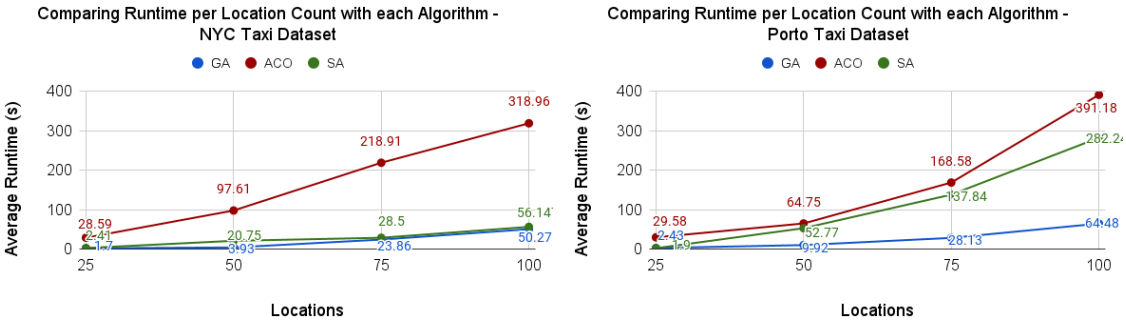



Figure 6.11: Average Runtime

between 0.1 and 30 units. On the other hand, the results obtained with Porto Dataset are more robust as the standard deviation range is only 3 units away from the mean. In conformity with algorithm reliability, GA had the highest standard deviation, ACO came in second and SA came in third. The gap between the ACO and SA standard deviation result was not enormous, especially with large instances such as with 75 and 100 locations. The results obtained are illustrated in Figure 6.12.

The second metric used to measure the reliability of the algorithm is the standard deviation of the average runtime on both datasets with the 4 problem instances. The standard deviation values were significantly high in the Porto Dataset in contrast to NYC Dataset. As shown in Figure 6.13, SA displayed the highest values in both datasets while GA and ACO had almost the same values in the NYC dataset but ACO was higher than GA in Porto dataset. This indicates that there was a large variation in the computation time of the experiments in each of the 5 trials with SA.

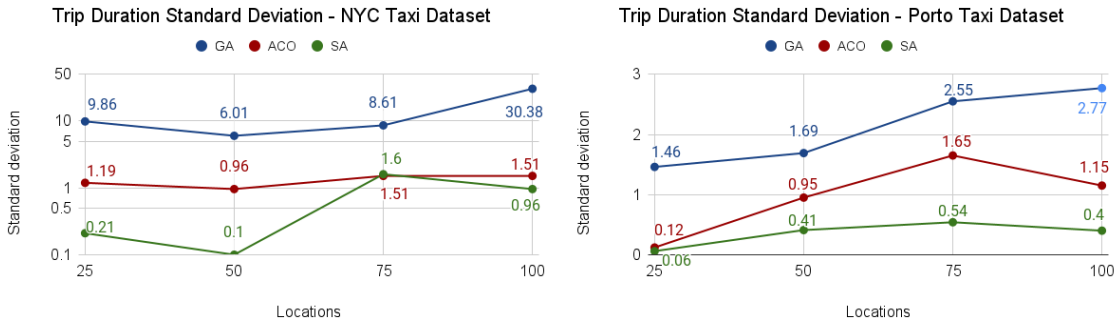



Figure 6.12: Trip Duration Standard Deviation

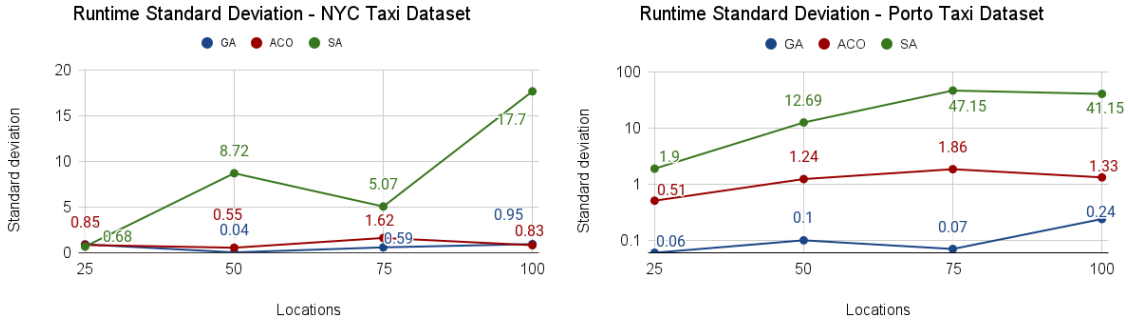



Figure 6.13: Runtime Standard Deviation

### 6.5.3 *Experimental Conclusions*

GA is the simplest algorithm to implement in comparison to ACO and SA using the scikit-opt library. Only the population size parameter needs to be adjusted, therefore the hurdle of multiple and parallel parameter tuning such as in ACO and in SA is avoided. The GA is the most cost efficient algorithm in terms of computation time and its runtime also does not vary largely each time the algorithm is executed. However, this comes at a cost of a very poor performance, and this is owing to multiple factors. The random selection of population in the beginning of the process is the main reason behind the poor solution quality due to the infamous issue with premature convergence. Moreover, as mentioned in chapter 5 before, the way the mutation operator works in this library limits the algorithm's ability to explore the solution space and improve the result while maintaining a high fitness score. The algorithm is found to be more suitable either with small problem instances or with large instances where the computation efficiency is prioritized over solution quality.

The main challenge encountered with implementing ACO is its sensitivity to parameter variation and the difficulty in tuning the three interdependent parameters; alpha, beta and rho at the same time. The ACO gives relatively good results in small- to medium-sized instances such as with 25 and 50 locations. Despite its outpaced performance compared to GA, ACO is susceptible to getting trapped in local minima because its pheromone is updated according to the current best path. However, giving

enough time for the pheromones to evaporate increases the probability of finding better results. Moreover, the pheromone evaporation mechanism counteracts the impact of the randomized initial solution which is usually far from optimal and responsible for converging towards a local minimum, unlike GA where the initial solution is generated in a purely random way. Nevertheless, the algorithm exhibits an unstable convergence behaviour as fluctuations take place along the whole search process till last iteration. This is due to having two main factors influencing path planning in each step, specifically the pheromone levels and the heuristic importance. Therefore, ACO does not necessarily converge towards an optimal and it is very common for the algorithm to find the optimal solution in the early iteration stages. It is clear that the optimal balance between them is difficult to find in each iteration, hence the continuous exploratory behaviour and the fluctuations. However, the library records the best solution found in all iterations and returns it.

SA has provided the most accurate and reliable results all over the different problem instances. It has found the optimal path with the shortest trip duration with slight differences in the obtained results in each of the five test runs. However, it is unstable in terms of computation time as the time needed to obtain the results varies greatly each time the algorithm is executed, especially with large problem instances. This is owing to its convergence behaviour, since after converging and staying stable, a small change can take place and the algorithm is then forced to iterate for longer time to ensure that no further improvements to the result are made. This mechanism however is the main driving force for the high solution quality, as well as the long and unstable execution time.

# 7 CONCLUSION AND FUTURE WORK

## 7.1 CONCLUSION

This thesis provides a comprehensive explanation of VRP and its significance in route planning for various vehicle routing applications. VRP can be modified and adapted to solve numerous optimisation problems by setting up constraints that reflect real-life problem specifications. It has a crucial role in transportation network planning which can be segmented into smaller problems simulating different routing scenarios. Some of these examples include applying VRP to manage a fleet of vehicles by planning the route with the least fuel consumption and in freight transportation where vehicle number, capacities, and delivery time windows constrict the problem. However, it is essential to first address the optimisation aspect of the VRP by bringing the TSP to the front line where the route with the least weight constraint is calculated. The main problem in TSP is computing an optimal solution in a realistic time without sacrificing much of the solution accuracy. The TSP can therefore be considered as the simplest version of the VRP as it involves only a single vehicle with no problem constraints.

Nevertheless, TSP can be generalised to solve other optimisation problems in other domains that are not solely related to vehicle routing such as in astronomy where astronomers tend to minimise time spent moving telescopes to observation sources or in DNA sequencing in biology, where a map of genomes is constructed to understand how DNA markers across the genome are related.

More on VRP as well as TSP and the challenges faced in finding optimal solutions are outlined. Then, traditional approaches employed over the past decades and their classifications to solve optimisation problems are briefly described and their limitations are highlighted to lay the ground for adopting meta-heuristics.

The experiments to compare the performance of the different meta-heuristics were carried out on two datasets: NYC Green Taxi dataset and Porto City Taxi dataset where they were first pre-processed and prepared for the experiments. Secondly, a machine learning model to forecast travel time was developed. These values were then used as an input for executing the experiments to find the shortest travel time and sequence of addresses to visit for a given unordered set of locations with the three meta-heuristics.

Then experiments were carried out on both datasets using GA, ACO and SA optimisation algorithms with different problem instances. The algorithms were evaluated based on the results' accuracy, reliability and efficiency in terms of computation time. The results from both datasets have led us to conclude that SA came first in providing the most accurate and most reliable results, where ACO came second. On the other hand, GA demonstrated the highest efficiency in finding the solutions, yet the results fell short in terms of accuracy and reliability.

## 7.2 FUTURE WORK

Meta-heuristics reduce computation complexity solving the TSP. However, the challenge of applying more constraints to the TSP to match real-world route planning problems, especially in different domains still stands.

For short term research, expanding the simple TSP scenario by applying more constraints such as time window and vehicle capacity can be tested using meta-heuristics. In addition, modifying the problem constraints to further test it on different problem domains is of an interest.

In the long term, we intend to test hybridising optimisation approaches whether by combining two meta-heuristics to solve the TSP or one meta-heuristic and one heuristic to compensate their performance shortcomings such as by using a heuristic in getting the initial solution and then applying a meta-heuristic to improve this solution.

# BIBLIOGRAPHY


[AGL07] J. E. Aarnes, T. Gimse, and K.-A. Lie. "An Introduction to the Numerics of Flow in Porous Media using Matlab". In: *Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF*. Ed. by G. Hasle, K.-A. Lie, and E. Quak. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 265–306. ISBN: 978-3-540-68783-2. DOI: 10.1007/978-3-540-68783-2_9. URL: https://doi.org/10.1007/978-3-540-68783-2_9.

[AH20] A. E. Adewale and A. Hadachi. "Neural Networks Model for Travel Time Prediction Based on ODTravel Time Matrix". In: *CoRR* abs/2004.04030 (2020). arXiv: 2004.04030. URL: https://arxiv.org/abs/2004.04030.

[Ade+12] A. Adewole, K. Otubamowo, T. Egunjobi, and K. Ng. "A Comparative Study of Simulated Annealing and Genetic Algorithm for Solving the Travelling Salesman Problem". In: *International Journal of Applied Information Systems* 4 (Oct. 2012), pp. 6–12. DOI: 10.5120/ijais12-450678.

[Alt16] L. Altenberg. "Evolutionary Computation". In: *Encyclopedia of Evolutionary Biology*. Ed. by R. M. Kliman. Oxford: Academic Press, 2016, pp. 40–47. ISBN: 978-0-12-800426-5. DOI: https://doi.org/10.1016/B978-0-12-800049-6.00307-3. URL: https://www.sciencedirect.com/science/article/pii/B9780128000496003073.

[Ang+01] S. Angus, M. Holger, F. W, K Phang, H Seah, and C Tan. "Selection of Parameters for Ant Colony Optimisation Applied to the Optimal Design of Water Distribution Systems". In: Dec. 2001.

[ASK17] A. M. S. Asih, B. M. Sopha, and G. Kriptaniadewa. "Comparison study of metaheuristics: Empirical application of delivery problems". In: *International Journal of Engineering Business Management* 9 (2017), p. 18479790177436o3. DOI: 10.1177/1847979017743603.

[Bac+00] Backer, Bruno, Furnon, Vincent, Shaw, Paul, Kilby, Philip, Prosser, and Patrick. "Solving Vehicle Routing Problems Using Constraint Programming and Metaheuristics". In: *J. Heuristics* 6 (Sept. 2000), pp. 501–523. DOI: 10.1023/A:1009621410177.

[Bak20] C. Bakshi. *Support vector regression*. 2020. URL: https://medium.com/swlh/support-vector-regression-explained-for-beginners-2a8d14ba6e5d#:~:text=While%20linear%20regression%20models%20minimize,called%20the%20epsilon%2Dinsensitive%20tube..

[Bay+13] Bayram, Hüsamettin, Şahin, and Ramazan. "A new simulated annealing approach for the traveling salesman problem". In: *Mathematical and Computational Applications* 18 (Dec. 2013), pp. 313–322. DOI: 10.3390/mca18030313.

[Ber+07] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. "Static pickup and delivery problems: a classification scheme and survey". In: *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research* 15.1 (2007), pp. 1–31. URL: https://EconPapers.repec.org/RePEc:spr:topjnl:v:15:y:2007:i:1:p:1-31.

[BCL10] G. Berbeglia, J.-F. Cordeau, and G. Laporte. "Dynamic pickup and delivery problems". In: *European Journal of Operational Research* 202.1 (2010), pp. 8–15. URL: https://EconPapers.repec.org/RePEc:eee:ejores:v:202:y:2010:i:1:p:8-15.

[BJM19] D. Bertsimas, P. Jaillet, and S. Martin. "Online Vehicle Routing: The Edge of Optimization in Large-Scale Applications". In: *Operations Research* 67.1 (2019), pp. 143–162. DOI: 10.1287/opre.2018.1763.

[Bos20] T. Bosona. "Urban Freight Last Mile Logistics—Challenges and Opportunities to Improve Sustainability: A Literature Review". In: *Sustainability* 12.21 (2020). ISSN: 2071-1050. DOI: 10.3390/su12218769. URL: https://www.mdpi.com/2071-1050/12/21/8769.

[BRN15] K. Braekers, K. Ramaekers, and I. Nieuwenhuyse. "The Vehicle Routing Problem: State of the Art Classification and Review". In: *Computers & Industrial Engineering* 99 (Dec. 2015). DOI: 10.1016/j.cie.2015.12.007.

[Bro21] J. Brownlee. *A gentle introduction to XGBoost for applied machine learning*. 2021. URL: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/.

[BG05] O. Bräysy and M. Gendreau. "Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms". In: *Transportation Science* 39 (Feb. 2005), pp. 104–118. DOI: 10.1287/trsc.1030.0056.

[COCS20] J. Cao, M. Olvera-Cravioto, and M. Shen. "Last-Mile Shared Delivery: A Discrete Sequential Packing Approach". In: *Mathematics of Operations Research* 45 (Nov. 2020), pp. 1466–1497. DOI: 10.1287/moor.2019.1039.

[CG16] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, 785–794. ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145/2939672.2939785.

[CF21] Z. Chen and W. Fan. "A Freeway Travel Time Prediction Method Based on an XGBoost Model". In: *Sustainability* 13 (July 2021), p. 8577. DOI: 10.3390/su13158577.

[CT16] R. Chris and L. Tanya. "Collective behaviour and swarm intelligence in slime moulds". In: *FEMS Microbiology Reviews* 40 (Aug. 2016), fuw033. DOI: 10.1093/femsre/fuw033.

[Cor+03] Cordeau, Jean-François, Laporte, and Gilbert. "The Dial-a-Ride Problem (DARP): Variants, modeling issues and algorithms". In: *Quarterly Journal of the Belgian, French and Italian Operations Research Societies* (2003). DOI: 10.1007/s10288-002-0009-8.

[Cor+07] J.-F. Cordeau, G. Laporte, M. Savelsbergh, and D. Vigo. "Vehicle Routing". In: vol. 14. Jan. 2007, pp. 195–224.

[DR59] G. B. Dantzig and J. H. Ramser. "The Truck Dispatching Problem". In: *Management Science* 6.1 (1959), pp. 80–91. DOI: 10.1287/mnsc.6.1.80.

[DW11] C. Darwin and A. Wallace. "On the Tendency of Species to form Varieties; and on the Perpetuation of Varieties and Species by Natural Means of Selection". In: *Zoological Journal of the Linnean Society* 3.9 (July 2011), pp. 45–62. ISSN: 0024-4082. DOI: 10.1111/j.1096-3642.1858.tb02500.x. eprint: https://academic.oup.com/zoolinnean/article-pdf/3/9/45/16610000/j.1096-3642.1858.tb02500.x.pdf. URL: https://doi.org/10.1111/j.1096-3642.1858.tb02500.x.

[Des+20] H. Desai, B. Remer, B. Chalaki, L. Zhao, A. Malikopoulos, and J. Rios-Torres. *Vehicle Routing for the Last-Mile Logistics Problem*. 2020. arXiv: 1904.05464 [math.OC].

[Des+15] S. Desale, A. Rasool, S. Andhale, and P. Rane. "Heuristic and Meta-Heuristic Algorithms and Their Relevance to the Real World: A Survey". In: *INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING IN RESEARCH TRENDS* 351 (Jan. 2015), pp. 2349–7084.

[Doe+19] J. Doering, R. Kizys, A. A. Juan, Àngels Fitó, and O. Polat. "Metaheuristics for rich portfolio optimisation and risk management: Current state and future trends". In: *Operations Research Perspectives* 6 (2019), p. 100121. ISSN: 2214-7160. DOI: https://doi.org/10.1016/j.orp.2019.100121. URL: https://www.sciencedirect.com/science/article/pii/S2214716019300399.

[DBS06] M. Dorigo, M. Birattari, and T. Stutzle. "Ant colony optimization". In: *IEEE Computational Intelligence Magazine* 1.4 (2006), pp. 28–39. DOI: 10.1109/MCI.2006.329691.

[DG97] M. Dorigo and L. M. Gambardella. "Ant colonies for the travelling salesman problem". In: *Biosystems* 43.2 (1997), pp. 73–81. ISSN: 0303-2647. DOI: https://doi.org/10.1016/S0303-2647(97)01708-5. URL: https://www.sciencedirect.com/science/article/pii/S0303264797017085.

[FO11] E. Filip and M. Otakar. "The travelling salesman problem and its application in logistic practice". In: 8 (Oct. 2011), pp. 163–173.

[GTA99] L. M. Gambardella, Éric Taillard, and G. Agazzi. "MACS-VRPTW: A Multiple Colony System For Vehicle Routing Problems With Time Windows". In: *New Ideas in Optimization*. McGraw-Hill, 1999, pp. 63–76.

[Gao21] W. Gao. "Modified ant colony optimization with improved tour construction and pheromone updating strategies for traveling salesman problem". In: *Soft Computing* 25 (Feb. 2021), pp. 1–27. DOI: 10.1007/s00500-020-05376-8.

[GBC15] A. Gharib, J. Benhra, and M. Chaouqi. "A performance comparison of PSO and GA applied to TSP". In: *International Journal of Computer Applications* *–No.* (Oct. 2015), pp. 975–8887. DOI: 10.5120/ijca2015907188.

[Gol88] D. E. Goldberg. "Genetic Algorithms in Search Optimization and Machine Learning". In: 1988.

[Gol+80] B. Golden, L. Bodin, T. Doyle, and W. Stewart. "Approximate Traveling Salesman Algorithms". In: *Operations Research* 28.3-part-ii (1980), pp. 694–711. DOI: 10.1287/opre.28.3.694. eprint: https://doi.org/10.1287/opre.28.3.694. URL: https://doi.org/10.1287/opre.28.3.694.

[Gup+18] B. Gupta, S. Awasthi, R. Gupta, L. Ram, P. Kumar, B. R. Prasad, and S. Agarwal. "Taxi Travel Time Prediction Using Ensemble-Based Random Forest and Gradient Boosting Model". In: 2018.

[Gup13] D. Gupta. "Solving TSP Using Various Meta-Heuristic Algorithms". In: *International Journal of Recent Contributions from Engineering, Science & IT (iJES)* 1 (Nov. 2013), p. 22. DOI: 10.3991/ijes.v1i2.3233.

[HCK12] A. Hajjam, J.-C. Créput, and A. Koukam. "From the TSP to the Dynamic VRP: An Application of Neural Networks in Population Based Metaheuristic". In: vol. 433. Jan. 2012, pp. 309–337. ISBN: 978-3-642-30664-8. DOI: 10.1007/978-3-642-30665-5.

[HI17] H. Halim and I. Ismail. "Combinatorial Optimization: Comparison of Heuristic Algorithms in Travelling Salesman Problem". In: *Archives of Computational Methods in Engineering* 26 (Nov. 2017). DOI: 10.1007/s11831-017-9247-y.

[Hol84] J. H. Holland. "Genetic Algorithms and Adaptation". In: *Adaptive Control of Ill-Defined Systems*. Ed. by O. G. Selfridge, E. L. Rissland, and M. A. Arbib. Boston, MA: Springer US, 1984, pp. 317–333. ISBN: 978-1-4684-8941-5. DOI: 10.1007/978-1-4684-8941-5_21. URL: https://doi.org/10.1007/978-1-4684-8941-5_21.

[Kan+19] K. Kankanamge, Y. Witharanage, C. Withanage, M. Hansini, D. Lakmal, and U. Thayasivam. "Taxi Trip Travel Time Prediction with Isolated XGBoost Regression". In: July 2019, pp. 54–59. DOI: 10.1109/MERCon.2019.8818915.

[Kar+16] A. Karevan, M. Homayouni, A. Hesami, and S. Larki. "The comparison between Simulated Annealing Algorithm and Genetic Algorithm in order to solve Traveling Salesman Problem in Esfahan telecommunications companies". In: Dec. 2016.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing". In: *Science* 220.4598 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671. eprint: https://www.science.org/doi/pdf/10.1126/science.220.4598.671. URL: https://www.science.org/doi/abs/10.1126/science.220.4598.671.

[Kol15] E. Kolog. *Dynamic Programming using Brute Force Algorithm for a Traveling Salesman Problem*. June 2015. DOI: 10.13140/RG.2.1.2304.2727.

[Kon20] G. D. Konstantakopoulos. "Vehicle routing problem and related algorithms for logistics distribution: a literature review and classification". In: *Operational Research* (2020).

[Kue20] I. Kuehner. "Swarm Intelligence for Solving a Traveling Salesman Problem". In: *eKNOW 2020, The Twelfth International Conference on Information, Process, and Knowledge Management*. 2020. URL: https://elib.dlr.de/136847/.

[LZ05] H.-E. LIN and R. Zito. "A review of travel-time prediction in transport and logistics". In: *Proceedings of the Eastern Asia Society for Transportation Studies* 5 (Jan. 2005).

[Lap09] G. Laporte. "Fifty Years of Vehicle Routing". In: *Transp. Sci.* 43 (2009), pp. 408–416.

[LQS20] C. Lazo, F. Quiñónez, and F. Sandoya. "Comparative Study of Algorithms Metaheuristics Based Applied to the Solution of the Capacitated Vehicle Routing Problem". In: May 2020. ISBN: 978-1-83962-453-7. DOI: 10.5772/intechopen.91972.

[Li+14] B. Li, D. Krushinsky, H. Reijers, and T. Woensel, van. *The share-a-ride problem : people and parcels sharing taxis*. English. BETA publicatie : working papers. Technische Universiteit Eindhoven, 2014.

[Li+15] B. Li, D. Krushinsky, H. Reijers, and T. Van Woensel. "The Share-a-Ride Problem: People and Parcels Sharing Taxis". In: *European Journal of Operational Research* 238 (Dec. 2015). DOI: 10.1016/j.ejor.2014.03.003.

[LZ16] P. Li and H. Zhu. "Parameter Selection for Ant Colony Algorithm Based on Bacterial Foraging Algorithm". In: *Mathematical Problems in Engineering* 2016 (Dec. 2016), pp. 1–12. DOI: 10.1155/2016/6469721.

[Lin+12] Y. Lin, W. Li, F. Qiu, and H. Xu. "Research on Optimization of Vehicle Routing Problem for Ride-sharing Taxi". In: *Procedia - Social and Behavioral Sciences* 43 (2012). 8th International Conference on Traffic and Transportation Studies (ICTTS 2012), pp. 494–502. ISSN: 1877-0428. DOI: https://doi.org/10.1016/j.sbspro.2012.04.122. URL: https://www.sciencedirect.com/science/article/pii/S1877042812010038.

[LZL21] C. Liu, G. Zheng, and Z. J. Li. "Learning to Route via Theory-Guided Residual Network". In: *ArXiv* abs/2105.08279 (2021).

[LY13] C. Liu and J. Yu. "Multiple depots vehicle routing based on the ant colony with the genetic algorithm". In: *Journal of Industrial Engineering and Management* 6.4 (2013), pp. 1013–1026. ISSN: 2013-0953. DOI: 10.3926/jiem.747. URL: http://www.jiem.org/index.php/jiem/article/view/747.

[Loy19] J. Loy. *Neural Network Projects with Python : The Ultimate Guide to Using Python to Explore the True Power of Neural Networks Through Six Projects*. English. Packt Publishing, Limited, 2019.

[Mal17] V. Mallawaarachchi. *Introduction to Genetic Algorithms — Including Example Code*. 2017. URL: https://medium.com/towards-data-science/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3.

[MSM10] R. Matai, S. Singh, and M. Mittal. "Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches". In: Nov. 2010. ISBN: 978-953-307-426-9. DOI: 10.5772/12909.

[Met+53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. "Equation of State Calculations by Fast Computing Machines". In: *The Journal of Chemical Physics* 21.6 (1953), pp. 1087–1092. DOI: 10.1063/1.1699114. eprint: https://doi.org/10.1063/1.1699114. URL: https://doi.org/10.1063/1.1699114.

[MPG12] K. J. Miclaus, R. Pratt, and M. V. Galati. "The Traveling Salesman Traverses the Genome: Using SAS® Optimization in JMP® Genomics to build Genetic Maps". In: 2012.

[Mil+95] B. L. Miller, B. L. Miller, D. E. Goldberg, and D. E. Goldberg. "Genetic Algorithms, Tournament Selection, and the Effects of Noise". In: *Complex Systems* 9 (1995), pp. 193–212.

[MV17] S. Muthuraman and V. P. Venkatesan. "A Comprehensive Study on Hybrid Meta-Heuristic Approaches Used for Solving Combinatorial Optimization Problems". In: *2017 World Congress on Computing and Communication Technologies (WCCCT)* (2017), pp. 185–190.

[NKP12] S. Nanda Kumar and R. Panneerselvam. "A Survey on the Vehicle Routing Problem and Its Variants". In: *Intelligent Information Management* 04 (Jan. 2012). DOI: 10.4236/iim.2012.43010.

[NM16] L. Nicolas and S. J Moura. "Optimal Routing and Charging of Electric Ride-Pooling Vehicles in Urban Networks". PhD thesis. 2016. URL: https://escholarship.org/uc/item/4m5369gs.

[Nil03] C. Nilsson. "Heuristics for the Traveling Salesman Problem". In: (Jan. 2003).

[Ota+16] Ota, Masayo, Huy, Vo, Silva, Claudio, Freire, and Juliana. "STaRS: Simulating Taxi Ride Sharing at Scale". In: *IEEE Transactions on Big Data* PP (Nov. 2016), pp. 1–1. DOI: 10.1109/TBDATA.2016.2627223.

[Pil+13] V. Pillac, M. Gendreau, C. Guéret, and A. Medaglia. "A review of dynamic vehicle routing problems". In: *European Journal of Operational Research* 225 (Feb. 2013), 1–11. DOI: 10.1016/j.ejor.2012.08.015.

[Pot96] J.-Y. Potvin. "Genetic algorithms for the traveling salesman problem". In: *Annals of Operations Research* 63 (1996), pp. 337–370.

[QF21] B. Qiu and W. D. Fan. "Machine Learning Based Short-Term Travel Time Prediction: Numerical Results and Comparative Analyses". In: *Sustainability* 13.13 (2021). ISSN: 2071-1050. DOI: 10.3390/su13137454. URL: https://www.mdpi.com/2071-1050/13/13/7454.

[Ran+18] L. Ranieri, S. Digiesi, B. Silvestri, and M. Roccotelli. "A Review of Last Mile Logistics Innovations in an Externalities Cost Reduction Vision". In: *Sustainability* 10 (Mar. 2018), p. 782. DOI: 10.3390/su10030782.

[Ren+20] C. Ren, J. Wang, Y. You, and Y. Zhang. "Routing Optimization for Shared Electric Vehicles with Ride-Sharing". In: *Complexity* 2020 (2020), pp. 1–13. DOI: 10.1155/2020/9560135.

[RCS16] J.-P. Rodrigue, C. Comtois, and B. Slack. *The Geography of Transport Systems*. Dec. 2016, pp. 1–440. ISBN: 9781315618159. DOI: 10.4324/9781315618159.

[RSF13] O. Roeva, T. Slavov, and S. Fidanova. "Population-based vs. single point search meta-heuristics for a pid controller tuning". In: 1 (Jan. 2013), pp. 200–230. DOI: 10.4018/978-1-4666-4450-2.ch007.

[RSI77] D. Rosenkrantz, R. Stearns, and P. II. "An Analysis of Several Heuristics for the Traveling Salesman Problem". In: *SIAM J. Comput.* 6 (Sept. 1977), pp. 563–581. DOI: 10.1137/0206041.

[San+14] Santi, Paolo, Resta, et al. "Quantifying the benefits of vehicle pooling with shareability networks". In: *Proceedings of the National Academy of Sciences of the United States of America* 111 (Sept. 2014). DOI: 10.1073/pnas.1403657111.

[SB10a] N. Sariff and N. Buniyamin. "Genetic Algorithm Versus Ant Colony Optimization Algorithm - Comparison of Performances in Robot Path Planning Application." In: Jan. 2010, pp. 125–132.

[SB10b] N. Sariff and N. Buniyamin. "Genetic Algorithm Versus Ant Colony Optimization Algorithm - Comparison of Performances in Robot Path Planning Application." In: Jan. 2010, pp. 125–132.

[SG13] K. Sörensen and F. W. Glover. "Metaheuristics". In: *Encyclopedia of Operations Research and Management Science*. Ed. by S. I. Gass and M. C. Fu. Boston, MA: Springer US, 2013, pp. 960–970. ISBN: 978-1-4419-1153-7. DOI: 10.1007/978-1-4419-1153-7_1167. URL: https://doi.org/10.1007/978-1-4419-1153-7_1167.

[TY21] S.-Y. Tan and W.-C. Yeh. "The Vehicle Routing Problem: State-of-the-Art Classification and Review". In: *Applied Sciences* 11.21 (2021). ISSN: 2076-3417. DOI: 10.3390/app112110295. URL: https://www.mdpi.com/2076-3417/11/21/10295.

[Sog] *The last-mile delivery challenge*. URL: https://www.sogeti.com/explore/reports/the-last-mile-delivery-challenge/.

[VLM20] T. Vidal, G. Laporte, and P. Matl. "A concise guide to existing and emerging vehicle routing problem variants". In: *European Journal of Operational Research* 286.2 (2020), pp. 401–416. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2019.10.010. URL: https://www.sciencedirect.com/science/article/pii/S0377221719308422.

[VMF04] M. Vittorio, G. L. Maria, and L. Fabio. "Ant Colony Optimization". In: (May 2004). DOI: 10.1007/978-3-540-39930-8_5.

[Wan+09] Wang, Zicheng, Geng, Xiutang, and Z. Shao. "An Effective Simulated Annealing Algorithm for Solving the Traveling Salesman Problem". In: *Journal of Computational and Theoretical Nanoscience* 6 (July 2009), pp. 1680–1686. DOI: 10.1166/jctn.2009.1230.

[WZL18] F. Wang, Y. Zhu, and J. Liu. "Ridesharing as a Service: Exploring Crowdsourced Connected Vehicle Information for Intelligent Package Delivery". In: June 2018, pp. 1–10. DOI: 10.1109/IWQoS.2018.8624152.

[Wu+03] C.-H. Wu, C.-C. Wei, D.-C. Su, M.-H. Chang, and J.-M. Ho. "Travel time prediction with support vector regression". In: Nov. 2003, 1438 –1442 vol.2. ISBN: 0-7803-8125-4. DOI: 10.1109/ITSC.2003.1252721.

[Yue+19] Yuen, C. Fai, Singh, A. Pratap, Goyal, Sagar, Ranu, Sayan, Bagchi, and Amitabha. "Beyond Shortest Paths: Route Recommendations for Ride-Sharing". In: *The World Wide Web Conference*. WWW '19. San Francisco, CA, USA: Association for Computing Machinery, 2019, 2258–2269. ISBN: 9781450366748. DOI: 10.1145/3308558.3313465. URL: https://doi.org/10.1145/3308558.3313465.

[ZZ17] L. Zhang and C. Zhan. "Machine Learning in Rock Facies Classification: An Application of XGBoost". In: May 2017, pp. 1371–1374. DOI: 10.1190/IGC2017-351.

[Zha+22] X. Zhang, L. Lauber, H. Liu, J. Shi, M. Xie, and Y. Pan. "Travel time prediction of urban public transportation based on detection of single routes". In: *PLOS ONE* 17.1 (Jan. 2022), pp. 1–23. DOI: 10.1371/journal.pone.0262535.

[ZL] P. Q. Zhao and P. Y. Liu. *Lecture notes on Introduction to Meta-heuristics*.

# A

# MORE RESULTS

## A.1 TABULATED RESULTS

Table A.1: GA - NYC Dataset Parameter tuning

| Locations | Pop. size | Generations | Mutation factor | Runtime (s) | Trip dur. (m) |
|---|---|---|---|---|---|
| 25 | 10 | 600 | 1 | 0.9 | 311.31 |
| 25 | 20 | 600 | 1 | 2.07 | 304.09 |
| 25 | 30 | 600 | 1 | 5.42 | 321.35 |
| 25 | 40 | 600 | 1 | 3.06 | 311.02 |
| 25 | 50 | 600 | 1 | 4.11 | 314.91 |
| 25 | 60 | 600 | 1 | 7.54 | 310.16 |
| 25 | 70 | 600 | 1 | 8.4 | 314.34 |
| 25 | 80 | 600 | 1 | 6.67 | 320.89 |
| 25 | 90 | 600 | 1 | 7.88 | 314.93 |
| 25 | 100 | 600 | 1 | 8.99 | 317.26 |
| 50 | 10 | 800 | 1 | 1.33 | 506.92 |
| 50 | 20 | 800 | 1 | 2.58 | 507.64 |
| 50 | 30 | 800 | 1 | 3.82 | 492.75 |
| 50 | 40 | 800 | 1 | 4.97 | 495.98 |
| 50 | 50 | 800 | 1 | 6.27 | 502.28 |
| 50 | 60 | 800 | 1 | 7.45 | 503.15 |
| 50 | 70 | 800 | 1 | 8.66 | 516.54 |
| 50 | 80 | 800 | 1 | 9.94 | 501.53 |
| 50 | 90 | 800 | 1 | 11.11 | 501.94 |
| 50 | 100 | 800 | 1 | 12.36 | 513.46 |
| 75 | 10 | 1200 | 1 | 4.16 | 717.28 |
| 75 | 20 | 1200 | 1 | 7.05 | 708.56 |
| 75 | 30 | 1200 | 1 | 10.57 | 699.14 |
| 75 | 40 | 1200 | 1 | 11.82 | 727.38 |
| 75 | 50 | 1200 | 1 | 14.99 | 720.24 |
| 75 | 60 | 1200 | 1 | 18.14 | 714.81 |
| 75 | 70 | 1200 | 1 | 20.56 | 715.94 |
| 75 | 80 | 1200 | 1 | 29.6 | 688.73 |
| 75 | 90 | 1200 | 1 | 30.97 | 684.3 |
| 75 | 100 | 1200 | 1 | 34.23 | 695.9 |
| 100 | 50 | 2000 | 1 | 29.31 | 894.96 |
| 100 | 60 | 2000 | 1 | 32.75 | 888.65 |
| 100 | 70 | 2000 | 1 | 41.51 | 894.05 |
| 100 | 80 | 2000 | 1 | 44.96 | 890.4 |

**Table A.1 continued from previous page**

| Locations | Pop. size | Generations | Mutation factor | Runtime (s) | Trip dur. (m) |
|---|---|---|---|---|---|
| 100 | 90 | 2000 | 1 | 50.45 | 880.05 |
| 100 | 100 | 2000 | 1 | 55.89 | 906.76 |
| 100 | 110 | 2000 | 1 | 61.54 | 891.32 |
| 100 | 120 | 2000 | 1 | 66.2 | 908.56 |
| 100 | 130 | 2000 | 1 | 71.77 | 894.73 |
| 100 | 140 | 2000 | 1 | 76.86 | 897.84 |

Table A.2: GA - Porto Dataset Parameter tuning

| Locations | Pop. size | Generations | Mutation factor | Runtime (s) | Trip duration (m) |
|---|---|---|---|---|---|
| 25 | 10 | 600 | 1 | 0.83 | 100.95 |
| 25 | 20 | 600 | 1 | 2.32 | 101.84 |
| 25 | 30 | 600 | 1 | 4.48 | 100 |
| 25 | 40 | 600 | 1 | 2.39 | 94.65 |
| 25 | 50 | 600 | 1 | 5.57 | 100.61 |
| 25 | 60 | 600 | 1 | 4.62 | 99.84 |
| 25 | 70 | 600 | 1 | 4.92 | 98.15 |
| 25 | 80 | 600 | 1 | 5.15 | 95.52 |
| 25 | 90 | 600 | 1 | 7.58 | 98.6 |
| 25 | 100 | 600 | 1 | 6.92 | 97.17 |
| 50 | 10 | 800 | 1 | 1.35 | 208.98 |
| 50 | 20 | 800 | 1 | 3.86 | 208.1 |
| 50 | 30 | 800 | 1 | 4.91 | 204.33 |
| 50 | 40 | 800 | 1 | 4.97 | 204.95 |
| 50 | 50 | 800 | 1 | 6.31 | 205.97 |
| 50 | 60 | 800 | 1 | 9.4 | 205.85 |
| 50 | 70 | 800 | 1 | 8.72 | 201.09 |
| 50 | 80 | 800 | 1 | 10.05 | 201.06 |
| 50 | 90 | 800 | 1 | 11.14 | 206.7 |
| 50 | 100 | 800 | 1 | 15.09 | 205.83 |
| 75 | 50 | 1200 | 1 | 14.5 | 327.96 |
| 75 | 60 | 1200 | 1 | 15.21 | 326.4 |
| 75 | 70 | 1200 | 1 | 21.39 | 320.14 |
| 75 | 80 | 1200 | 1 | 20.75 | 318.98 |
| 75 | 90 | 1200 | 1 | 24.08 | 319.62 |
| 75 | 100 | 1200 | 1 | 26.47 | 324.72 |
| 75 | 110 | 1200 | 1 | 28.04 | 315.66 |
| 75 | 120 | 1200 | 1 | 32.36 | 316.09 |
| 75 | 130 | 1200 | 1 | 33.18 | 315.99 |
| 75 | 140 | 1200 | 1 | 35.95 | 320.49 |
| 100 | 50 | 2000 | 1 | 27.27 | 424.14 |
| 100 | 60 | 2000 | 1 | 34.25 | 416.55 |
| 100 | 70 | 2000 | 1 | 39.52 | 438.84 |

**Table A.2 continued from previous page**

| Locations | Pop. size | Generations | Mutation factor | Runtime (s) | Trip duration (m) |
|---|---|---|---|---|---|
| 100 | 80 | 2000 | 1 | 44.72 | 424.77 |
| 100 | 90 | 2000 | 1 | 49.21 | 421.96 |
| 100 | 100 | 2000 | 1 | 54.26 | 413.69 |
| 100 | 110 | 2000 | 1 | 60.71 | 422.14 |
| 100 | 120 | 2000 | 1 | 64.75 | 411.35 |
| 100 | 130 | 2000 | 1 | 71 | 420.29 |
| 100 | 140 | 2000 | 1 | 75.61 | 423.95 |

Table A.3: ACO - NYC Dataset Parameter tuning with locations = 50, population size = 30 and generations = 800

| alpha | beta | NYC Runtime (s) | NYC Trip duration (m) |
|---|---|---|---|
| 0.5 | 0.5 | 102.79 | 869.1 |
| 0.5 | 1 | 102.21 | 626.89 |
| 0.5 | 2 | 102.31 | 537.44 |
| 0.5 | 3 | 101.46 | 519.32 |
| 1 | 0.5 | 100.61 | 543.36 |
| 1 | 1 | 100.67 | 511.78 |
| 1 | 2 | 100.67 | 508.19 |
| 1 | 3 | 101.23 | 506.15 |
| 1 | 5 | 99.86 | 507.1 |
| 1.2 | 0.5 | 103.22 | 536.32 |
| 1.2 | 1 | 100.48 | 509.02 |
| 1.2 | 2 | 100.45 | 512.71 |
| 1.2 | 3 | 101.1 | 508.84 |
| 1.2 | 5 | 98.89 | 508.15 |
| 2 | 0.5 | 100.36 | 551.99 |
| 2 | 1 | 100.07 | 541.65 |
| 2 | 2 | 100.28 | 513.84 |
| 2 | 3 | 101.06 | 513.2 |
| 2 | 5 | 99.48 | 512.51 |
| 5 | 0.5 | 100.47 | 557.53 |
| 5 | 1 | 100.33 | 532.79 |
| 5 | 2 | 101.26 | 521.44 |
| 5 | 3 | 100.04 | 514.78 |
| 5 | 5 | 100.94 | 509.18 |

Table A.4: ACO - NYC Dataset Parameter tuning with locations = 50, population size = 30 and generations = 800

| alpha | beta | NYC Runtime (s) | NYC Trip duration (m) |
|---|---|---|---|
| 1 | 2 | 99.58 | 509.75 |
| 1 | 2 | 99.78 | 505.64 |

**Table A.4 continued from previous page**

| alpha | beta | NYC Runtime (s) | NYC Trip duration (m) |
|---|---|---|---|
| 1 | 2 | 100.37 | 506.64 |
| 1 | 3 | 100.56 | 507.06 |
| 1 | 3 | 100.18 | 505.49 |
| 1 | 3 | 100.97 | 505.25 |
| 1 | 5 | 102.19 | 504.23 |
| 1 | 5 | 100.34 | 507.7 |
| 1 | 5 | 99.37 | 505.64 |
| 1.2 | 5 | 101.45 | 506.42 |
| 1.2 | 5 | 99.07 | 506.97 |
| 1.2 | 5 | 101.3 | 506.47 |

Table A.5: ACO - NYC Dataset Parameter tuning

| Locations | Pop. size | Generations | alpha | beta | Runtime (s) | Trip duration (m) |
|---|---|---|---|---|---|---|
| 25 | 10 | 800 | 1 | 5 | 15.58 | 310.12 |
| 25 | 20 | 800 | 1 | 5 | 31.45 | 308.65 |
| 25 | 30 | 800 | 1 | 5 | 48.09 | 308.92 |
| 25 | 40 | 800 | 1 | 5 | 62.04 | 308.08 |
| 25 | 50 | 800 | 1 | 5 | 79.9 | 307.54 |
| 25 | 60 | 800 | 1 | 5 | 95.21 | 307.79 |
| 25 | 70 | 800 | 1 | 5 | 107.29 | 307.82 |
| 25 | 80 | 800 | 1 | 5 | 123.84 | 307.82 |
| 25 | 90 | 800 | 1 | 5 | 131.91 | 307.48 |
| 25 | 100 | 800 | 1 | 5 | 147.98 | 308.76 |
| 50 | 10 | 800 | 1 | 5 | 34.15 | 512.26 |
| 50 | 20 | 800 | 1 | 5 | 70.97 | 507.92 |
| 50 | 30 | 800 | 1 | 5 | 101.08 | 507.31 |
| 50 | 40 | 800 | 1 | 5 | 137.75 | 508.04 |
| 50 | 50 | 800 | 1 | 5 | 170.5 | 508.04 |
| 50 | 60 | 800 | 1 | 5 | 201.92 | 507.25 |
| 50 | 70 | 800 | 1 | 5 | 238.4 | 507.46 |
| 50 | 80 | 800 | 1 | 5 | 269.61 | 506.9 |
| 50 | 90 | 800 | 1 | 5 | 302.08 | 508.08 |
| 50 | 100 | 800 | 1 | 5 | 332.63 | 507.65 |
| 75 | 10 | 800 | 1 | 5 | 57.69 | 696.34 |
| 75 | 20 | 800 | 1 | 5 | 110.24 | 694.87 |
| 75 | 30 | 800 | 1 | 5 | 163.39 | 695.58 |
| 75 | 40 | 800 | 1 | 5 | 215.06 | 692.03 |
| 75 | 50 | 800 | 1 | 5 | 287.73 | 692.98 |
| 75 | 60 | 800 | 1 | 5 | 331.05 | 692.91 |
| 75 | 70 | 800 | 1 | 5 | 389.34 | 694.18 |
| 75 | 80 | 800 | 1 | 5 | 469.8 | 693.25 |
| 75 | 90 | 800 | 1 | 5 | 511.49 | 694.04 |

**Table A.5 continued from previous page**

| Locations | Pop. size | Generations | alpha | beta | Runtime (s) | Trip duration (m) |
|---|---|---|---|---|---|---|
| 75 | 100 | 800 | 1 | 5 | 557.12 | 693.18 |
| 100 | 10 | 800 | 1 | 5 | 81.4 | 908.76 |
| 100 | 20 | 800 | 1 | 5 | 163.06 | 904.95 |
| 100 | 30 | 800 | 1 | 5 | 243.38 | 902.44 |
| 100 | 40 | 800 | 1 | 5 | 325.61 | 896.69 |
| 100 | 50 | 800 | 1 | 5 | 408.2 | 899.24 |
| 100 | 60 | 800 | 1 | 5 | 490.4 | 902.44 |
| 100 | 70 | 800 | 1 | 5 | 571.86 | 900.45 |
| 100 | 80 | 800 | 1 | 5 | 641.09 | 895.18 |
| 100 | 90 | 800 | 1 | 5 | 725.87 | 896.73 |
| 100 | 100 | 800 | 1 | 5 | 799.11 | 894.69 |

Table A.6: ACO - Porto Dataset Parameter tuning with locations = 50, population size = 30 and generations = 800

| alpha | beta | Runtime (s) | Trip duration (m) |
|---|---|---|---|
| 0.5 | 0.5 | 106.71 | 250.28 |
| 0.5 | 1 | 102.27 | 239.54 |
| 0.5 | 2 | 102.38 | 217.93 |
| 0.5 | 3 | 103.64 | 212.49 |
| 1 | 0.5 | 101.61 | 215.28 |
| 1 | 1 | 101.45 | 204.75 |
| 1 | 2 | 102.55 | 201.91 |
| 1 | 3 | 102.45 | 199.02 |
| 1 | 5 | 102.79 | 198.03 |
| 1.2 | 0.5 | 101.58 | 208.3 |
| 1.2 | 1 | 103.73 | 197.77 |
| 1.2 | 2 | 103.74 | 197.8 |
| 1.2 | 3 | 103.77 | 196.9 |
| 1.2 | 5 | 102.27 | 198.55 |
| 2 | 0.5 | 101 | 222.34 |
| 2 | 1 | 100.42 | 205.32 |
| 2 | 2 | 101.82 | 201.16 |
| 2 | 3 | 102.69 | 196.77 |
| 2 | 5 | 101.5 | 197.71 |
| 5 | 0.5 | 102.41 | 230.38 |
| 5 | 1 | 102.18 | 212.6 |
| 5 | 2 | 103.87 | 201.46 |
| 5 | 3 | 105.9 | 198.89 |
| 5 | 5 | 103.69 | 200.37 |

Table A.7: ACO - Porto Dataset Parameter tuning with locations = 50, population size = 30 and generations = 800

| **alpha** | **beta** | **Runtime (s)** | **Trip duration (m)** |
|---|---|---|---|
| 1.2 | 1 | 104.58 | 197.5 |
| 1.2 | 1 | 102.69 | 200.81 |
| 1.2 | 1 | 103.04 | 200.09 |
| 1.2 | 3 | 101.23 | 198.6 |
| 1.2 | 3 | 102.62 | 198.88 |
| 1.2 | 3 | 102.05 | 199.32 |
| 2 | 3 | 105.88 | 198.06 |
| 2 | 3 | 103.11 | 197.98 |
| 2 | 3 | 102.78 | 197.15 |
| 2 | 5 | 107.45 | 197.74 |
| 2 | 5 | 103.54 | 196.42 |
| 2 | 5 | 103.56 | 196.42 |

Table A.8: ACO - Porto Dataset Parameter tuning

| **Locations** | **Pop. size** | **Generations** | **alpha** | beta | Runtime (s) | Trip duration (m) |
|---|---|---|---|---|---|---|
| 25 | 10 | 800 | 2 | 5 | 15.43 | 95.35 |
| 25 | 20 | 800 | 2 | 5 | 29.62 | 94.46 |
| 25 | 30 | 800 | 2 | 5 | 43.27 | 94.98 |
| 25 | 40 | 800 | 2 | 5 | 56.89 | 94.75 |
| 25 | 50 | 800 | 2 | 5 | 72.04 | 94.58 |
| 25 | 60 | 800 | 2 | 5 | 84.39 | 94.54 |
| 25 | 70 | 800 | 2 | 5 | 100.15 | 94.31 |
| 25 | 80 | 800 | 2 | 5 | 114.78 | 93.65 |
| 25 | 90 | 800 | 2 | 5 | 129.8 | 93.49 |
| 25 | 100 | 800 | 2 | 5 | 142.66 | 94.54 |
| 50 | 10 | 800 | 2 | 5 | 33.11 | 197.99 |
| 50 | 20 | 800 | 2 | 5 | 66.16 | 194.21 |
| 50 | 30 | 800 | 2 | 5 | 99.93 | 196.18 |
| 50 | 40 | 800 | 2 | 5 | 133.1 | 197.12 |
| 50 | 50 | 800 | 2 | 5 | 164.64 | 194.49 |
| 50 | 60 | 800 | 2 | 5 | 203.06 | 194.67 |
| 50 | 70 | 800 | 2 | 5 | 232.89 | 194.46 |
| 50 | 80 | 800 | 2 | 5 | 263.04 | 193.89 |
| 50 | 90 | 800 | 2 | 5 | 296.25 | 195.3 |
| 50 | 100 | 800 | 2 | 5 | 331.97 | 194.26 |
| 75 | 10 | 800 | 2 | 5 | 53.29 | 308.12 |
| 75 | 20 | 800 | 2 | 5 | 109.24 | 306.47 |
| 75 | 30 | 800 | 2 | 5 | 166.4 | 304.02 |
| 75 | 40 | 800 | 2 | 5 | 220.51 | 306.05 |
| 75 | 50 | 800 | 2 | 5 | 274.66 | 307.16 |

**Table A.8 continued from previous page**

| Locations | Pop. size | Generations | alpha | beta | Runtime (s) | Trip duration (m) |
|---|---|---|---|---|---|---|
| 75 | 60 | 800 | 2 | 5 | 331.92 | 305.18 |
| 75 | 70 | 800 | 2 | 5 | 384.12 | 307.17 |
| 75 | 80 | 800 | 2 | 5 | 438.65 | 304.83 |
| 75 | 90 | 800 | 2 | 5 | 499.77 | 304.37 |
| 75 | 100 | 800 | 2 | 5 | 572.38 | 305.25 |
| 100 | 10 | 800 | 2 | 5 | 76.83 | 399.96 |
| 100 | 20 | 800 | 2 | 5 | 180.65 | 397.45 |
| 100 | 30 | 800 | 2 | 5 | 289.02 | 395.09 |
| 100 | 40 | 800 | 2 | 5 | 389.79 | 396.6 |
| 100 | 50 | 800 | 2 | 5 | 390.77 | 395.17 |
| 100 | 60 | 800 | 2 | 5 | 594.96 | 397.12 |
| 100 | 70 | 800 | 2 | 5 | 700.04 | 395.92 |
| 100 | 80 | 800 | 2 | 5 | 811.94 | 396.16 |
| 100 | 90 | 800 | 2 | 5 | 914.63 | 395.55 |
| 100 | 100 | 800 | 2 | 5 | 1015.91 | 396 |

Table A.9: SA - NYC Dataset Parameter tuning : Tmax

| Locations | Tmax | Tmin | L = n*locations | Max. counter | Runtime (s) | Trip dur. (m) |
|---|---|---|---|---|---|---|
| 50 | 0.5 | 1.00E-01 | 10 | 150 | 2.03 | 499.4 |
| 50 | 1 | 1.00E-01 | 10 | 150 | 31.36 | 491.47 |
| 50 | 5 | 1.00E-01 | 10 | 150 | 9.51 | 511.55 |
| 50 | 10 | 1.00E-01 | 10 | 150 | 12.41 | 496.69 |
| 50 | 20 | 1.00E-01 | 10 | 150 | 13.67 | 507.17 |
| 50 | 30 | 1.00E-01 | 10 | 150 | 16.75 | 527.69 |
| 50 | 40 | 1.00E-01 | 10 | 150 | 15.18 | 550.87 |
| 50 | 50 | 1.00E-01 | 10 | 150 | 23.33 | 574.34 |

Table A.10: SA - NYC Dataset Parameter tuning : Tmin

| Loc. | Tmax | Tmin | L = n*locations | Max. counter | Runtime (s) | Trip dur. (m) |
|---|---|---|---|---|---|---|
| 50 | 1 | 3.00E-01 | 10 | 150 | 0.38 | 502.2 |
| 50 | 1 | 2.00E-01 | 10 | 150 | 2.07 | 500.3 |
| 50 | 1 | 1.00E-01 | 10 | 150 | 31.36 | 491.47 |
| 50 | 1 | 1.00E-02 | 10 | 150 | 19.74 | 498.73 |
| 50 | 1 | 1.00E-03 | 10 | 150 | 14.35 | 508.35 |
| 50 | 1 | 1.00E-04 | 10 | 150 | 19.36 | 491.49 |
| 50 | 1 | 1.00E-05 | 10 | 150 | 39.75 | 505.25 |
| 50 | 1 | 1.00E-06 | 10 | 150 | 14.87 | 498.45 |
| 50 | 1 | 1.00E-07 | 10 | 150 | 22.85 | 502.59 |
| 50 | 1 | 1.00E-08 | 10 | 150 | 24.76 | 491.51 |
| 50 | 1 | 1.00E-09 | 10 | 150 | 12.39 | 491.42 |
| 50 | 1 | 1.00E-10 | 10 | 150 | 19.46 | 491.5 |

**Table A.10 continued from previous page**

| Loc. | Tmax | Tmin | L = n*locations | Max. counter | Runtime (s) | Trip dur. (m) |
|---|---|---|---|---|---|---|
| 50 | 1 | 1.00E-10 | 10 | 150 | 25.92 | 498.72 |

Table A.11: SA - NYC Dataset Parameter tuning : L and Max. Counter

| Loc. | Tmax | Tmin | L = n*locations | Max. counter | Runtime (s) | Trip dur. (m) |
|---|---|---|---|---|---|---|
| 25 | 1 | 1.00E-09 | 10 | 100 | 2.48 | 309.24 |
| 25 | 1 | 1.00E-09 | 10 | 200 | 2.28 | 309.7 |
| 25 | 1 | 1.00E-09 | 10 | 300 | 3.1 | 310.59 |
| 25 | 1 | 1.00E-09 | 10 | 400 | 2.41 | 303.48 |
| 25 | 1 | 1.00E-09 | 10 | 500 | 2.77 | 307.17 |
| 25 | 1 | 1.00E-09 | 20 | 100 | 4.75 | 309.7 |
| 25 | 1 | 1.00E-09 | 20 | 200 | 4.58 | 307.17 |
| 25 | 1 | 1.00E-09 | 20 | 300 | 6.88 | 303.48 |
| 25 | 1 | 1.00E-09 | 20 | 400 | 4.42 | 310.59 |
| 25 | 1 | 1.00E-09 | 20 | 500 | 7.05 | 307.17 |
| 25 | 1 | 1.00E-09 | 40 | 100 | 10.46 | 303.48 |
| 25 | 1 | 1.00E-09 | 40 | 200 | 9.43 | 307.17 |
| 25 | 1 | 1.00E-09 | 40 | 300 | 8.29 | 310.59 |
| 25 | 1 | 1.00E-09 | 40 | 400 | 9 | 309.7 |
| 25 | 1 | 1.00E-09 | 40 | 500 | 10.16 | 307.17 |
| 50 | 1 | 1.00E-09 | 10 | 100 | 6.11 | 491.95 |
| 50 | 1 | 1.00E-09 | 10 | 200 | 31.39 | 491.42 |
| 50 | 1 | 1.00E-09 | 10 | 300 | 25.6 | 507.32 |
| 50 | 1 | 1.00E-09 | 10 | 400 | 55.22 | 508.03 |
| 50 | 1 | 1.00E-09 | 10 | 500 | 53.65 | 498.61 |
| 50 | 1 | 1.00E-09 | 20 | 100 | 14.58 | 498.81 |
| 50 | 1 | 1.00E-09 | 20 | 200 | 38.27 | 491.44 |
| 50 | 1 | 1.00E-09 | 20 | 300 | 34.34 | 491.48 |
| 50 | 1 | 1.00E-09 | 20 | 400 | 71.6 | 491.24 |
| 50 | 1 | 1.00E-09 | 20 | 500 | 77.95 | 491.33 |
| 50 | 1 | 1.00E-09 | 40 | 100 | 66.32 | 491.4 |
| 50 | 1 | 1.00E-09 | 40 | 200 | 38.87 | 500.01 |
| 50 | 1 | 1.00E-09 | 40 | 300 | 61.26 | 509.92 |
| 50 | 1 | 1.00E-09 | 40 | 400 | 65.87 | 491.51 |
| 50 | 1 | 1.00E-09 | 40 | 500 | 38.81 | 502.9 |
| 75 | 1 | 1.00E-09 | 10 | 100 | 25.31 | 679.7 |
| 75 | 1 | 1.00E-09 | 10 | 200 | 30.75 | 682.42 |
| 75 | 1 | 1.00E-09 | 10 | 300 | 18.85 | 680.86 |
| 75 | 1 | 1.00E-09 | 10 | 400 | 37.33 | 687.11 |
| 75 | 1 | 1.00E-09 | 10 | 500 | 21.52 | 679.28 |
| 75 | 1 | 1.00E-09 | 20 | 100 | 56.25 | 681.2 |
| 75 | 1 | 1.00E-09 | 20 | 200 | 58.9 | 680.79 |
| 75 | 1 | 1.00E-09 | 20 | 300 | 46.84 | 680.53 |

**Table A.11 continued from previous page**

| Loc. | Tmax | Tmin | L = n*locations | Max. counter | Runtime (s) | Trip dur. (m) |
|---|---|---|---|---|---|---|
| 75 | 1 | 1.00E-09 | 20 | 400 | 75.17 | 680.62 |
| 75 | 1 | 1.00E-09 | 20 | 500 | 28.83 | 679.51 |
| 75 | 1 | 1.00E-09 | 40 | 100 | 191.63 | 678.16 |
| 75 | 1 | 1.00E-09 | 40 | 200 | 74.81 | 678.31 |
| 75 | 1 | 1.00E-09 | 40 | 300 | 146.83 | 678.59 |
| 75 | 1 | 1.00E-09 | 40 | 400 | 131.37 | 679.52 |
| 75 | 1 | 1.00E-09 | 40 | 500 | 171.52 | 679.88 |
| 100 | 1 | 1.00E-09 | 10 | 100 | 71.6 | 881.84 |
| 100 | 1 | 1.00E-09 | 10 | 200 | 56.91 | 873.49 |
| 100 | 1 | 1.00E-09 | 10 | 300 | 53.36 | 873.53 |
| 100 | 1 | 1.00E-09 | 10 | 400 | 47.45 | 867.66 |
| 100 | 1 | 1.00E-09 | 10 | 500 | 96.53 | 876.59 |
| 100 | 1 | 1.00E-09 | 20 | 100 | 131.24 | 874.55 |
| 100 | 1 | 1.00E-09 | 20 | 200 | 84.25 | 906.45 |
| 100 | 1 | 1.00E-09 | 20 | 300 | 89.36 | 877.18 |
| 100 | 1 | 1.00E-09 | 20 | 400 | 118.82 | 893.91 |
| 100 | 1 | 1.00E-09 | 40 | 100 | 110.77 | 868.96 |
| 100 | 1 | 1.00E-09 | 40 | 200 | 108.13 | 876.53 |
| 100 | 1 | 1.00E-09 | 40 | 300 | 216.37 | 883.15 |
| 100 | 1 | 1.00E-09 | 40 | 400 | 128.28 | 879.35 |
| 100 | 1 | 1.00E-09 | 40 | 500 | 128.07 | 894.88 |

Table A.12: SA - Porto Dataset Parameter tuning : Tmax

| Loc. | Tmax | Tmin | L = n*locations | Max. counter | Runtime (s) | Trip dur. (m) |
|---|---|---|---|---|---|---|
| 50 | 0.5 | 1.00E-01 | 10 | 150 | 2.39 | 194.5 |
| 50 | 1 | 1.00E-01 | 10 | 150 | 28.38 | 192.66 |
| 50 | 5 | 1.00E-01 | 10 | 150 | 14.73 | 204.41 |
| 50 | 10 | 1.00E-01 | 10 | 150 | 24.53 | 216.45 |
| 50 | 20 | 1.00E-01 | 10 | 150 | 12.39 | 234.24 |
| 50 | 30 | 1.00E-01 | 10 | 150 | 12.5 | 237.71 |
| 50 | 40 | 1.00E-01 | 10 | 150 | 12.83 | 242.3 |
| 50 | 50 | 1.00E-01 | 10 | 150 | 10.33 | 243.04 |

Table A.13: SA - Porto Dataset Parameter tuning : Tmin

| Loc. | Tmax | Tmin | L = n*locations | Max. counter | Runtime (s) | Trip dur. (m) |
|---|---|---|---|---|---|---|
| 50 | 1 | 3.00E-01 | 10 | 150 | 1.11 | 198.37 |
| 50 | 1 | 2.00E-01 | 10 | 150 | 2.25 | 196.92 |
| 50 | 1 | 1.00E-01 | 10 | 150 | 17.35 | 193.36 |
| 50 | 1 | 1.00E-02 | 10 | 150 | 12.78 | 193.75 |
| 50 | 1 | 1.00E-03 | 10 | 150 | 16.79 | 193.14 |
| 50 | 1 | 1.00E-04 | 10 | 150 | 21.5 | 192.06 |

**Table A.13 continued from previous page**

| Loc. | Tmax | Tmin | L = n*locations | Max. counter | Runtime (s) | Trip dur. (m) |
|---|---|---|---|---|---|---|
| 50 | 1 | 1.00E-05 | 10 | 150 | 13.95 | 192.86 |
| 50 | 1 | 1.00E-06 | 10 | 150 | 12.59 | 192.99 |
| 50 | 1 | 1.00E-07 | 10 | 150 | 18.68 | 192.31 |
| 50 | 1 | 1.00E-08 | 10 | 150 | 13.73 | 193.79 |
| 50 | 1 | 1.00E-09 | 10 | 150 | 11.17 | 193.1 |
| 50 | 1 | 1.00E-10 | 10 | 150 | 9.08 | 194.05 |

Table A.14: SA - Porto dataset Parameter tuning : L and Max. counter

| Loc. | Tmax | Tmin | L = n*locations | Max. counter | Runtime (s) | Trip dur. (m) |
|---|---|---|---|---|---|---|
| 25 | 1 | 1.00E-04 | 10 | 100 | 2.58 | 93.6 |
| 25 | 1 | 1.00E-04 | 10 | 200 | 6.06 | 93.49 |
| 25 | 1 | 1.00E-04 | 10 | 300 | 6.4 | 93.49 |
| 25 | 1 | 1.00E-04 | 10 | 400 | 12.37 | 93.6 |
| 25 | 1 | 1.00E-04 | 10 | 500 | 10.34 | 93.6 |
| 25 | 1 | 1.00E-04 | 20 | 100 | 4.52 | 93.49 |
| 25 | 1 | 1.00E-04 | 20 | 200 | 8.77 | 93.49 |
| 25 | 1 | 1.00E-04 | 20 | 300 | 21.73 | 93.49 |
| 25 | 1 | 1.00E-04 | 20 | 400 | 12.46 | 93.49 |
| 25 | 1 | 1.00E-04 | 20 | 500 | 18.41 | 93.49 |
| 25 | 1 | 1.00E-04 | 40 | 100 | 9.89 | 93.49 |
| 25 | 1 | 1.00E-04 | 40 | 200 | 15.06 | 93.49 |
| 25 | 1 | 1.00E-04 | 40 | 300 | 26.24 | 93.49 |
| 25 | 1 | 1.00E-04 | 40 | 400 | 28.34 | 93.49 |
| 25 | 1 | 1.00E-04 | 40 | 500 | 36.22 | 93.49 |
| 50 | 1 | 1.00E-04 | 10 | 100 | 9.71 | 193.59 |
| 50 | 1 | 1.00E-04 | 10 | 200 | 14.58 | 192.98 |
| 50 | 1 | 1.00E-04 | 10 | 300 | 25.99 | 192.66 |
| 50 | 1 | 1.00E-04 | 10 | 400 | 34.33 | 191.76 |
| 50 | 1 | 1.00E-04 | 10 | 500 | 56.52 | 192.12 |
| 50 | 1 | 1.00E-04 | 20 | 100 | 33.28 | 192.34 |
| 50 | 1 | 1.00E-04 | 20 | 200 | 57.49 | 191.91 |
| 50 | 1 | 1.00E-04 | 20 | 300 | 54.92 | 191.83 |
| 50 | 1 | 1.00E-04 | 20 | 400 | 140.09 | 191.53 |
| 50 | 1 | 1.00E-04 | 20 | 500 | 59.38 | 192.23 |
| 50 | 1 | 1.00E-04 | 40 | 100 | 65.83 | 191.9 |
| 50 | 1 | 1.00E-04 | 40 | 200 | 74.09 | 191.93 |
| 50 | 1 | 1.00E-04 | 40 | 300 | 97.16 | 191.98 |
| 50 | 1 | 1.00E-04 | 40 | 400 | 100.43 | 192.27 |
| 50 | 1 | 1.00E-04 | 40 | 500 | 152.23 | 191.88 |
| 75 | 1 | 1.00E-04 | 10 | 100 | 22.62 | 300.48 |
| 75 | 1 | 1.00E-04 | 10 | 200 | 42.41 | 298.95 |
| 75 | 1 | 1.00E-04 | 10 | 300 | 100.6 | 298.15 |

**Table A.14 continued from previous page**

| Loc. | Tmax | Tmin | L = n*locations | Max. counter | Runtime (s) | Trip dur. (m) |
|---|---|---|---|---|---|---|
| 75 | 1 | 1.00E-04 | 10 | 400 | 41.15 | 299.48 |
| 75 | 1 | 1.00E-04 | 10 | 500 | 94.74 | 297.79 |
| 75 | 1 | 1.00E-04 | 20 | 100 | 32.46 | 300.74 |
| 75 | 1 | 1.00E-04 | 20 | 200 | 70.24 | 299.53 |
| 75 | 1 | 1.00E-04 | 20 | 300 | 166.03 | 297.6 |
| 75 | 1 | 1.00E-04 | 20 | 400 | 106.37 | 298.46 |
| 75 | 1 | 1.00E-04 | 20 | 500 | 221.92 | 297.95 |
| 75 | 1 | 1.00E-04 | 40 | 100 | 89.96 | 299.39 |
| 75 | 1 | 1.00E-04 | 40 | 200 | 244.43 | 298.02 |
| 75 | 1 | 1.00E-04 | 40 | 300 | 365.41 | 297.49 |
| 75 | 1 | 1.00E-04 | 40 | 400 | 378.68 | 297.32 |
| 75 | 1 | 1.00E-04 | 40 | 500 | 592.62 | 297.16 |
| 100 | 1 | 1.00E-04 | 10 | 100 | 35 | 393.73 |
| 100 | 1 | 1.00E-04 | 10 | 200 | 85.13 | 389.33 |
| 100 | 1 | 1.00E-04 | 10 | 300 | 111.44 | 390.166 |
| 100 | 1 | 1.00E-04 | 10 | 400 | 81.6 | 390.11 |
| 100 | 1 | 1.00E-04 | 10 | 500 | 165.47 | 390.09 |
| 100 | 1 | 1.00E-04 | 20 | 100 | 62.66 | 393.14 |
| 100 | 1 | 1.00E-04 | 20 | 200 | 145.48 | 389.275 |
| 100 | 1 | 1.00E-04 | 20 | 300 | 210.64 | 389.56 |
| 100 | 1 | 1.00E-04 | 20 | 400 | 302.65 | 389.53 |
| 100 | 1 | 1.00E-04 | 20 | 500 | 333.5 | 389.25 |
| 100 | 1 | 1.00E-04 | 40 | 100 | 184.97 | 389.69 |
| 100 | 1 | 1.00E-04 | 40 | 200 | 253.67 | 389.54 |
| 100 | 1 | 1.00E-04 | 40 | 300 | 749.92 | 388.32 |
| 100 | 1 | 1.00E-04 | 40 | 400 | 398.32 | 389.84 |